The Dissertation Committee for Rishab Goyal
certifies that this is the approved version of the following dissertation:

Collusion Resistant Traitor Tracing Systems

Committee:

Brent R. Waters, Supervisor

Adam R. Klivans

Hovav Shacham

Daniel Wichs

Collusion Resistant Traitor Tracing Systems

by

Rishab Goyal

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2019

Dedicated to my grandparents (ADMRG).

Acknowledgments

My pursuit towards a doctoral degree has been an incredible journey that lead to a significant intellectual enrichment as well as personal and professional development. Here I wish to thank the multitudes of people who helped me in reaching this wonderful milestone. I sincerely apologize to anyone who I unintentionally forget to acknowledge.

First and foremost, I would like to express my immense gratitude to my advisor, Brent Waters. He has been an instrumental force throughout my graduate studies, and the results obtained in this dissertation are in-part a result of his continuous tutelage and insightful comments over the course of my PhD. I am also grateful to Venkata Koppula, Willy Quach, Andrew Russell, Satyanarayana Vusirikala, Daniel Wichs for their numerous discussions regarding the problems related to the concept of traitor tracing, and especially Venkata, Willy, and Daniel for their fruitful collaborations that led to the results obtained as part of this dissertation. A special thank you goes to Venkata (B.L.) for teaching me a lot of things, and being a wonderful friend and colleague. He has been a major player in my fun-filled grad school life, and was always available which lead to many illuminating conversations (both technical and non-technical).

I also would like to thank all my past advisors and research men-

tors (Vipul Goyal, Ragesh Jaiswal, Raghav Bhaskar, Dhruv Mahajan, Sanjiva Prasad, Matteo Maffei, Michael Backes) as well as the CS faculties at UT and IIT Delhi for sharing their vast wealth of knowledge. I am grateful to Adam Klivans, Hovav Shacham, and Daniel for being a part of my PhD committee, and Emmett Witchel and David Zuckerman for being a part of my RPE committee. I would like to particularly thank Allison, Amit, Hoeteck, Ryo, Shweta, Vinod, Zvika for their inspiring discussions on technical topics, and all the fellow students and colleagues across the globe that I had the pleasure to interact with over the course of my doctoral studies (Shashank, Mubashir, Ghufran, Swadhin, Satya, David, Sam, Divya, Prabhanjan, Dakshita, Aayush, Saikrishna, Ashutosh, Giorgos, Willy, Sebastian).

A special thanks to Melinda (Lindy) Aleshire for being a wonderful person to interact with, and taking care of all the administrative requirements. I am also very grateful to all the funding agencies (especially IBM for the PhD fellowship) that have supported my research over the last several years, as well as all the administrative staff at UT and all other places I have visited for research related activities. My gratitude goes to all my friends throughout the years for their endless support and guidance, especially my Beatles group (Rahul, Bharat, Abdul) and Google trio (Tushar, Mayank, Ashwin).

Last but most importantly and significantly, I wish to acknowledge everyone in my family for their consistent and unconditional support and love. Especially my parents, sister, and brother-in-law (AANRG) for their insurmountable confidence and belief in my capabilities, and a constant reminder

of the most valuable things in life. Finally, I dedicate this dissertation to my late grandparents (ADMRG) without whom none of this was possible.

Collusion Resistant Traitor Tracing Systems

Publication No. _____

Rishab Goyal, Ph.D.
The University of Texas at Austin, 2019

Supervisor: Brent R. Waters

The notion of traitor tracing was introduced by Chor, Fiat, and Naor [47] in the early 90s with the goal of solving accountability problem in broadcast systems. While the original motivation was of catching users that create pirate decoder boxes in broadcast TV systems, there are several applications that go beyond that setting. Despite wide applicability of such systems all existing solutions for the traitor tracing problem suffer from one or more deficiencies (such as have large ciphertext size, or prove security in the bounded-collusion setting, or provide only weak tracing guarantees, or rely on strong cryptographic assumptions).

In this proposed thesis, we provide the first traitor tracing scheme that does not suffer from any of the aforementioned deficiencies. That is, we provide a collusion resistant traitor tracing system with ciphertexts that grow polynomially in $\log(n)$ (where n is the number of users), and prove it secure under the Learning with Errors (LWE) assumption [107, 108]. This is the first

traitor tracing scheme with such parameters provably secure from a standard cryptographic assumption.

We achieve our results by first conceiving a novel approach to building traitor tracing that starts with a new form of Functional Encryption that we call Mixed FE. In a Mixed FE system the encryption algorithm is bimodal and works with either a public key or master secret key. Ciphertexts encrypted using the public key can only encrypt one type of functionality. On the other hand the secret key encryption can be used to encode many different types of programs, but is only secure as long as the attacker sees a bounded number of such ciphertexts.

We first show how to combine Mixed FE with Attribute-Based Encryption to achieve traitor tracing. Second we build Mixed FE systems for polynomial sized branching programs (which corresponds to the complexity class LOGSPACE) by relying on the polynomial hardness of the LWE assumption with super-polynomial modulus-to-noise ratio. In addition to achieving new traitor tracing results, the techniques developed push forward the broader area of computing on encrypted data under standard assumptions. Notably, traitor tracing is a substantially different problem from other cryptography primitives that have seen recent progress in LWE-based solutions.

Additionally, we show that our techniques could be used to provide new constructions for traitor tracing systems with embedded identity tracing functionality [104] as well.

Table of Contents

List of Figures

Chapter 1

Introduction

The concept of covert communication, or *encryption*, can be traced to as early as the era of ancient empires such as the Roman Empire (see Caesar cipher [38]). This predates the conception of computers itself by many centuries. Even until modern times, the broad science of cryptography was referred almost exclusively to as encryption. Simply put, encryption can be understood as the process of recasting sensitive information (called plaintext) into an unintelligible form (called ciphertext). While decryption is the opposite process of recovering the plaintext from a ciphertext.

Until about forty years ago, a widely-held belief was that for two parties to establish a private communication channel, that is covertly communicate, it was necessary for them to share some common secret information (called a secret key). This belief was dispelled by Diffie and Hellman [57, 58] who put forth an unconventional and radical concept of public-key cryptography, where two parties can covertly communicate with each other without sharing any a-priori agreed upon mutual secret.

Since its advent, public-key encryption (PKE) [57, 58, 109, 73] has remained a cornerstone in modern-day cryptography and has been one of the

most widely used and studied cryptographic primitive. It is an invaluable tool and its use is ubiquitous in building tools from secure web communication (e.g., SSH, SSL), to disk encryption, and secure software patch distribution to state a few. The traditional view of public-key encryption, however, only enables a one-to-one private communication channel between two users over a public broadcast network. Over the last several decades, significant research effort has been made by the cryptographic community in re-envisioning the original goals of public-key encryption, in turn pushing towards more expressiveness from such systems thereby enabling newer applications. This effort has lead to the introduction of encryption systems with more powerful functionalities such as identity-based encryption (IBE) [114, 49, 20], broadcast encryption (BE) [66, 16], searchable encryption (SE) [18], attribute-based encryption (ABE) [111, 86], and most notably functional encryption (FE) [112, 26] that is meant to encapsulate all previously thought-out PKE, IBE, BE, SE, and ABE functionalities.

The problem

Even with such re-envisioning, the fundamental underlying goal of encryption systems has stayed the same, that is to enable secure communication and deliver content. This leaves open the possibility of an entirely different and uncaptured attack strategy which can succinctly be labeled as *piracy*. For instance, consider a scenario where the content provider (say AT&T, Netflix etc.) wants to privately broadcast to a set of *paying* customers. Now if an

attacker either corrupts some of the already paying customers, or buys a limited number of subscriptions, then it can potentially extract secret information from the corrupt recipient devices, and use it to produce arbitrary pirate devices that recover the private content. Such pirate devices could be extremely detrimental for the content provider (i.e., the sender), and although this constitues as unauthorized distribution of private content, the regular plaintext hiding encryption systems do not provide any guarantees against such attackers.

Motivated by the piracy problem in widescale content distribution, Chor, Fiat, and Naor [47] in the early 90s, introduced the concept of traitor tracing. Traitor tracing (TT) systems are designed especially to prevent such adversarial threats. Intuitively, these systems provide the content distributor the capability to "trace" the customers who are responsible for the unauthorized distribution as long as the content distributor can get access to one such pirate device. Formally, the notion of traitor tracing is captured as follows.

In a (traitor) tracing [47] system an authority runs a setup algorithm that takes in a security parameter λ and the number, n, of users in the system. The setup outputs a public key pk, master secret key msk, and n secret keys $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$. The system has an encryption algorithm that uses the public key pk to create a ciphertext for a message m that is decryptable by any of the n secret keys, but where the message will be hidden from any user who does not have access to the keys. The last property is the standard plaintext hiding, more commonly referred to as the IND-CPA security (or, indistinguishability

under chosen plaintext attack). The most salient feature of a traitor tracing system is the presence of an additional "tracing algorithm" which is used to identify corrupt/coerced users. Suppose an attacker corrupts some subset $S \subseteq \{1, \ldots, n\}$ of authorized users and produces a special decryption box D that can decrypt the ciphertexts with some nonnegligible probability. The tracing property of the system states that the tracing algorithm, given the master secret and *only* oracle access to box D, outputs a set of users T where T contains at least one user from the colluding set S and no users outside of S.

The landscape

Existing approaches for achieving traitor tracing can be fit in the framework of private linear broadcast encryption (PLBE) introduced by Boneh, Sahai, and Waters (BSW) [25]. In a PLBE system the setup algorithm takes as input a security parameter λ and the number of users n. It outputs a public key pk, master secret key msk, and n private keys $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$ where a user with index j is given key sk_j. Any of the private keys is capable of decrypting a ciphertext ct created using pk. However, there is an additional $\mathsf{TrEncrypt}$ algorithm that takes in the master secret key, a message, and an index i. This produces a ciphertext that only users with index $j > i$ can decrypt. Moreover, any adversary-produced decryption box D that was created with private keys in the set of S will not be able to distinguish between encryptions to index $i - 1$ or i, where $i \notin S$. In addition, encryptions of two different messages

m_0, m_1 to index n must be indistinguishable.

The tracing system is set up by simply running the PLBE setup and distributing each PLBE key to the corresponding user. To trace the set of colluding parties given a decoding box D, the tracing algorithm first measures (with several samples) the probability that D correctly decrypts a ciphertext encrypted to index i for all $i \in [0, n]$. If the box D originally decrypted with probability ϵ, then there must exist some index i where the probability the box decrypts on index $i - 1$ is at least ϵ/n more than the probability it decrypts on ciphertexts encrypted to index i, since by PLBE security D cannot decrypt encryptions to index n with nonnegligible probability. At this point the tracing algorithm marks user i as a traitor/colluder.

Prior to developments made as part of this thesis, there were three approaches to building PLBE. The most basic approach is to simply create n public/private key pairs under a standard IND-CPA secure public key encryption system. A PLBE ciphertext is formed by encrypting the message m to each user's public key individually and concatenating all of the subciphertexts to form one long ciphertext, $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_n)$. A user with secret key sk_i in the system will decrypt by running decryption on ct_i and ignore the rest of the ciphertext components. To $\mathsf{TrEncrypt}$ to index i simply encrypt the all 0's string in first i ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_i$ in place of the message. The index hiding property follows directly from IND-CPA security of the underlying encryption scheme, since without secret key i no attacker can determine whether ct_i is an encryption of the message or all 0's string.

The above approach works because there is a portion of the ciphertext ct_i *dedicated* to each user i in the system which is not touched during the decryption process of other users with keys sk_j for $j \neq i$. This dedicated ciphertext space strategy makes it easy to silently kill user i's ability to access the message in a way unnoticeable to other users, but also inherently requires a ciphertext size that grows linearly in n. In order to achieve PLBE with sublinear size ciphertexts, one needs to implement some form of computing on encrypted data.

BSW [25] provided the first construction that achieved PLBE with ciphertext growth that was sublinear in n. They leveraged composite order bilinear groups to achieve ciphertexts that grew proportionally to $\sqrt{n}$. While future variants [27, 70, 67] used bilinear maps to obtain additional properties, the ciphertext size for all bilinear map based constructions remained stuck at the $\sqrt{n}$ mark.

Several years later Boneh and Zhandry [29] showed how to utilize indistinguishability obfuscation [11, 12] and apply punctured programming techniques [113] to achieve the ideal case where ciphertexts grow polynomially in $\log(n)$ and λ. The downside of applying indistinguishability obfuscation is that all current obfuscation candidates are based on nonstandard multilinear map group assumptions, and several such multilinear candidates [68, 54, 71, 56] have been attacked (see [55, 45, 51, 31, 52, 53, 28, 89, 87, 44, 98, 46, 7] and the references therein). (One could also achieve similar results by using the functional encryption scheme of Garg et al. [69], but this also relies on multilinear

maps.)

Therefore, even with all this progress the following question remained unresolved since the inception of traitor tracing in early 90s:

Can we build secure traitor tracing with $\text{POLY}(\lambda, \log(n))$*-sized ciphertexts from standard assumptions?*

The result

In this thesis we resolve the above question by providing a traitor tracing scheme with ciphertexts that grow polynomially in $\log(n)$ and λ and prove it secure under the learning with errors (LWE) assumption [107, 108]. This is the first traitor tracing scheme with such parameters that is provably secure from a standard assumption.[1] In addition to achieving new traitor tracing results, we believe our techniques push forward the broader area of computing on encrypted data under standard assumptions. Notably, traitor tracing is a substantially different problem from other cryptography primitives that have seen recent progress in LWE solutions.

We achieve our result by first conceiving a novel approach to building traitor tracing that starts with a new form of functional encryption that we call mixed functional encryption (Mixed FE). In a Mixed FE system the encryption algorithm is bimodal and works with either a public key or master secret

[1] The results provided here were obtained in a joint work with Venkata Koppula and Brent Waters [82, 83].

key. Ciphertexts encrypted using the public key can only encrypt one type of functionality. On the other hand, the secret key encryption can be used to encode many different types of programs, but is only secure as long as the attacker sees a bounded number of such ciphertexts.

We first show how to combine Mixed FE with attribute-based encryption (ABE) to achieve traitor tracing. Second, we show under the LWE assumption how to construct Mixed FE systems for polynomial-sized branching programs (which corresponds to the complexity class LOGSPACE). A detailed description of our techniques is provided in the next chapter in Section 2.1.

Towards more applications

While the concept of traitor tracing was originally motivated by catching corrupt users in broadcast systems, the notion of traitor tracing has numerous other applications such as transmitting sensitive information to first responders (or military personnel etc) on an ad-hoc deployed wireless network, accessing and sharing encrypted files on untrusted cloud storage etc. This propels us to study the problem of traitor traitor more finely with a dedicated focus on understanding the issues that prevent a wider adoptability of such systems.

One major hurdle is that, as per the traditional description of the problem, the tracing portion (that is identifying the corrupt users) is inherently tied to the central authority (key generator) in the system. This is due to the fact that the authority needs to keep track of the users who have been

issued private keys, and thus it needs to maintain an explicit mapping (as a look-up table) between the user identification information and the indices of their respective private keys. Otherwise, the output of the tracing algorithm will simply be a subset T of the user indices which can not be linked to actual users in the system, thereby introducing the problem of accountability and circumventing the whole point of tracing traitors. In addition, this not only constrains the authority to be fully stateful (with the state size growing linear with the number of users) by necessitating that the authority *must record* the user information to key index mapping, but also restricts the authority to be the only party which can perform any meaningful notion of tracing if (authorized) user privacy/anonymity is also desired.[2] Therefore, even if the TT system achieves public traceability, that is the tracing key key can be included as part of public parameters, no third party would be able to identify traitors in system due to lack of a public mapping as described above.

Furthermore, in certain situations the user information to key index mapping might be undetermined. For example, suppose all the users in the system obtain their private decryption keys without revealing any sensitive identification information to the key generating authority. (Note that this can be achieved by some sort of two party computation-based transfer between the user and authority.) In such a scenario, it is not clear how tracing would

[2]Although the problem of statefulness can be avoided by posting the identity of all authorized users along with their respective (decryption key) indices on a public-bulletin board, such a solution is particularly undesirable in practice as the user identities might include highly sensitive information such as passport information, driving license number, etc.

work since the authority would not be able to point to any user in the system as a traitor because the key index to user identity mapping is unknown, even if the tracing algorithm correctly outputs an index of some coerced secret key.

Embedding identities

These observations lead to the following question —

Is it possible to embed the user identification information in the private decryption keys such that during tracing the algorithm not only finds the corrupted key indices, but also extracts the corresponding user identities from the pirate decoding box?

Formally, this is captured by giving an additional parameter κ as an input to the setup algorithm, where κ denotes the length of the user identities that can be embedded in the private keys. The setup now outputs a master secret key msk, instead of n private user keys, where msk is used to generate private keys $\mathsf{sk}_{i,\mathsf{id}}$ for any index-identity pair $(i, \mathsf{id}) \in [n] \times \{0,1\}^\kappa$. And the tracing algorithm outputs a set of user identities $T \subseteq \{0,1\}^\kappa$, where $\mathsf{id} \in T$ indicates that id was one of the corrupted users.[3] This interpretation of traitor tracing resolves the above issues of statefulness, third-party traceability, and maintaining a private look-up table for providing user anonymity.

[3]Note that the tracing algorithm could be additionally asked to output the corresponding user index along with the identity, but since the index $i \in [n]$ could itself be encoded in the identity id using only $\log(n)$ bits therefore this seems unnecessary.

The above-stated question of traitor tracing with embedded information in secret keys was first studied by Nishimaki, Wichs, and Zhandry [104]. Their approach was to directly work with the existing private linear broadcast encryption (PLBE) framework [25], however that resulted in solutions based on non-standard assumptions. Concretely, they assume existence of an adaptively-secure collusion-resistant public-key functional encryption (FE) scheme with compact ciphertexts. Currently all known instantiations are either based on multilinear maps [68, 54, 71, 56], or indistinguishability obfuscation [11, 12].

An important open question here is whether the above problem of embedded information traitor tracing can be solved from standard assumptions. In this thesis, we study this question as well and provide a general framework for solving the identity embedding problem.

Reinforcing the main result

Augmenting our solution to the regular traitor tracing problem, we also provide new constructions for traitor tracing systems with embedded identity tracing functionality.[4] We build a *compact* embedded identity traitor tracing schemes (EITT) scheme secure under the learning with errors (LWE) assump-

[4]Nishimaki, Wichs, and Zhandry [104] used the term "flexible" traitor tracing to refer to schemes where the space of identities that can be traced is exponential. Here we call such TT systems as embedded identity traitor tracing schemes (or EITT for short).

tion.[5] Here compactness is defined analogous to regular TT, that is it means the size of ciphertexts and public key scales only polynomially with $\log(n)$ and κ, where n is the number of users and κ denotes the length of identities embedded. (Since one could set the number of users as exponential in the security parameter λ, thus one could simply interpret the size of all system parameters as scaling polynomially in λ and κ.)

Furthermore, a very important and useful piece of our approach is that it allows us to avoid subexponential security loss in the transformation (due to complexity leveraging) if we allow an *exponential* number of users in the system and the intermediate primitives used are only *selectively-secure*. Particularly, this is useful since our LWE-based solution relies on the Mixed FE primitive that we introduce, and currently we only provide a selective proof of security for our underlying Mixed FE scheme. Therefore, our approach also answers the question whether adaptivity is necessary for building EITT schemes if the system is required to support an unbounded number of users. Note that in the prior work of Nishimaki, Wichs, and Zhandry [104], it was crucial that they start with an "adaptively-secure" FE scheme for security purposes, but here our approach helps in bypassing the adaptivity requirement.[6]

A detailed description of our techniques for embedding identities is

[5]The results provided here were obtained in a joint work with Venkata Koppula and Brent Waters [84].

[6]Following our work, Chen et al. [42] provide two new constructions for Mixed FE that are secure under the LWE assumption. And, one of the proposed constructions were proven to be adaptively-secure.

provided in the next chapter in Section 2.2.

1.1 Impact of our results and techniques

In just a short duration (between now and the original publication of our traitor tracing construction [82, 83]) our techniques and framework have seen numerous other applications in making progress as well as resolving various other open problems. The results obtained in the domain of embedded identity traitor tracing [84], that partially contribute to this thesis, also serve as an indicator of the usefulness and wide applicability of our techniques. Below we highlight some recent results that are direct derivatives of our constructions and the techniques therein.

Following our work, Chen et al. [42] extended our traitor tracing framework to provide new constructions for the problem of attribute-based traitor tracing that are provably secure under the LWE assumption. Very briefly, an attribute-based traitor tracing system is a regular attribute-based encryption system in which the predicate key also contains a user tag, and the tracing property of the system states that the authority can trace corrupt users from a useful decoding box as long as it can decrypt ciphertexts encrypted under any arbitrary selected attribute string. Additionally, Chen et al. presented two alternate constructions for Mixed FE, one from lockable obfuscation [80, 117] and the other starting from key-homomorphic private constrained pseudorandom functions [39, 36, 43].

Moreover, recently in two independent lines of work [78, 85], we were

able to demonstrate applications of our techniques in the problems of software watermarking [11, 99, 88, 119, 12, 103, 50] as well as broadcast and trace [102, 100]. Specifically, in [78] we presented the first construction for collusion resistant watermarkable attribute-based encryption schemes from standard assumptions. The starting point of our watermarking construction is also our Mixed FE framework. While in [85], we extended our Mixed FE abstraction by adding a broadcast feature which we called broadcast mixed functional encryption, and we showed that combining such broadcast mixed functional encryption schemes with attribute-based encryption with succinct ciphertexts, we obtained new results to the broadcast and trace problem from standard assumptions with significantly shorter ciphertexts than other existing broadcast and trace systems from standard assumptions [27].

As we partially elaborate in this thesis, in our most recent work [84] we show that our Mixed FE abstraction also has direct applications in embedding identities in a traitor tracing system. For building the embedding identity traitor tracing system, we relied on our Mixed FE scheme being highly expressive, that is instead of comparison functions we exploited the fact that our Mixed FE scheme supports encrypting a much richer class of functions (polynomial-sized branching programs). And most recently, Kim and Wu [91] studied the problem of trace and revoke again looking through the lens of Mixed FE.

In conclusion, all these follow-up works [42, 78, 85, 84, 91] support the belief that the techniques developed in this thesis have a much broader impact.

1.2 Organization

In Chapter 2, we provide a detailed overview of our techniques and discuss some future and related works. Next, in Chapter 3 we present the preliminaries required for this work, and in Chapter 4 we provide the definition for traitor tracing and introduce a new primitive called Mixed FE. In Chapter 5, we show how 1-query PLBE implies decoder-based PLBE, and how decoder-based PLBE suffices for constructing TT schemes. Therefore, the problem of constructing a TT scheme reduces to the problem of constructing a 1-query PLBE scheme, which we in turn show how to construct using Mixed FE and ABE. In Chapter 7, we present our Mixed FE construction (before presenting the Mixed FE construction, in Chapter 6 we present new lattice tools which are required for our construction). Lastly, in Chapter 8, we describe our approach for embedding identities in TT schemes. Also, in Chapter A we briefly describe an alternate TT construction that achieves perfect correctness as long as the underlying ABE schem is perfectly correct.

Chapter 2

Overview of our techniques

In this chapter, we provide a detailed overview of the techniques developed for building efficient collusion resistant traitor tracing systems that also provide the embedded identity tracing functionality. First, we overview our techniques for building a regular traitor tracing scheme that achieves the desired efficiency and collusion resistance properties. Next, we describe how our introduced framework could be extended to embed identities as well. Finally, we conclude the chapter by describing a few directions for future work as well as discussing some related works.

2.1 Building traitor tracing

We now give a technical overview of our main traitor tracing result. This overview is broken into four parts. In the first part we review the BSW notion of private linear broadcast encryption (PLBE) and its transformation into a traitor tracing system. Along the way we discover that the PLBE definitions as presented in [25] do *not* imply traitor tracing. We then show how to repair the argument by giving the attacker an additional oracle encryption query in the PLBE definitions. Second, we present the notion of Mixed FE

and show how an ABE and Mixed FE system (for the right functionalities) can be used to construct a PLBE system. The third part of our overview describes a new LWE toolkit which includes "enhanced" versions of lattice trapdoor sampling algorithms with additional security properties. Finally, we outline our main ideas for constructing the Mixed FE system and proving it secure under the LWE assumption.

2.1.1 Part 1: Breaking and Repairing the PLBE to Tracing Argument

First, let us review the PLBE algorithms as defined in [25]. A PLBE scheme consists of a setup, encryption, decryption, and trace-encryption algorithm. The setup algorithm outputs a public key, a master secret key, and n secret keys, one for each index in $[n]$. The encryption/decryption algorithms are self-explanatory; the trace-encryption algorithm is a special encryption algorithm that requires the master secret key, and can be used to encrypt a message to any index $i \in [0, n]$. The output ciphertext can be decrypted only by secret keys for indices $j > i$. BSW defined three security properties. The first security property (public to zero-index indistinguishability) requires that a standard encryption of message m is indistinguishable from a trace-encryption of m to the index 0, even when the adversary has all n secret keys. The second security property (index hiding) states that a trace-encryption of m to index $i - 1$ is indistinguishable from a trace-encryption of m to index i, even when the adversary has all the secret keys except the ith one. Finally, the third security property states that trace-encryption of m_0 to index n is indis-

tinguishable from trace-encryption of m_1 to index n, even when the adversary is given all n secret keys.

BSW argued that these three properties of PLBE are sufficient for constructing a traitor tracing (TT) scheme. In their transformation, the TT public key and n secret keys are set to be the PLBE public key and n secret keys, respectively. The TT encryption/decryption algorithms are identical to the PLBE encryption/decryption algorithms. Finally, the tracing algorithm uses the PLBE trace-encryption algorithm. Given a decoder box D, the tracing algorithm encrypts random messages to each index and checks if D can decrypt it correctly. If the decoder box is ϵ-successful[1] in decrypting (standard) encryptions, then it is also ϵ successful in decrypting trace-encryptions to index 0 (via the first security property). Next, note that the decoder box cannot decrypt trace-encryptions to index n (via the message indistinguishability property). Therefore, there must exist an index $i^* \in [n]$ where the success of the decoder box in decrypting trace-encryptions to index $i^* - 1$ is at least ϵ/n more than its success in decrypting trace-encryptions to i^*. This index i^* must be one of the indices queried by the adversary (since if the adversary does not have a key for index i^*, then the decoder box must not be able to distinguish between trace-encryptions to $i^* - 1$ and i^*). For each index i, the tracing algorithm computes an estimate of the decoder box's success probability in decrypting random trace-encryptions for index i. For all indices i where the

[1] A decoder box is said to be ϵ-successful if its probability of correctly decrypting a ciphertext is at least ϵ, where the probability is taken over the choice of the ciphertext and D's random coins.

measured success probabilities for $i-1$ and i are substantially different, user i is declared to be a traitor.

At an intuitive level, it seems like the BSW transformation should work. However, here we argue that it is indeed possible to have a PLBE scheme secure under the original BSW definition, but produce an insecure TT scheme in this regard. The problem lies in the fact that there is a "semantic gap" between the TT definition and the PLBE definition. The TT definition considers an attacker that produces a (stateless) decoder D whose success on decrypting multiple trace-encryptions is measured, whereas the PLBE definition considers indistinguishability on a single ciphertext (in particular, no ciphertext queries). Diving deeper, we show a separation by adding a feature to a PLBE scheme where the feature does not impact PLBE security, but results in an insecure TT scheme.

Given any secure PLBE scheme P, consider a scheme P' defined as follows. The setup algorithm of P' is similar to the setup of P, except it also samples an additional pseudorandom function (PRF) key K as part of the master secret key (we will assume the PRF has single-bit output). The (standard) encryption algorithm computes a ciphertext ct using the underlying scheme's encryption algorithm, chooses a uniformly random bit b, and outputs (ct, b). The trace-encryption of message m is the ciphertext $\mathrm{ct}' = (\mathrm{ct}, y = \mathrm{PRF}_K(i))$, where ct is the ciphertext obtained from the trace-encryption algorithm of P. It is easy to see that the new scheme satisfies all three PLBE security definitions, since there are no encryption queries allowed in the PLBE scheme

beyond the challenge ciphertext.

However, it is possible to construct a decoding box using only the secret key for index n such that the trace algorithm falsely accuses some user $i < n$. The decoder D, on input of a ciphertext $\mathsf{ct}' = (\mathrm{ct}, y)$, tests if $y = 1$. If so, it decrypts the ciphertext using key sk_n; otherwise it outputs a random message. Using PRF security, we can argue that there exists an index $i < n$ such that $\mathrm{PRF}_K(i-1) = 1$ and $\mathrm{PRF}_K(i) = 0$ with high probability. In this case the probability that D decrypts ciphertexts for index $i-1$ will be measurably different than the case in which it decrypts ciphertext for index i. Thus user i will be flagged as a colluder.

We repair the BSW transformation from PLBE to TT by considering a modified set of PLBE security definitions and prove that these do imply TT. We do so in two steps. First, we consider a *decoder-based* version of the BSW PLBE definitions. For concreteness, let us consider the index hiding definition. The decoder-based version of the index hiding version states that no adversary, given all secret keys except the ith one, can produce a decoder box D and a message m such that D can distinguish between trace-encryptions of m to index $i-1$ and trace-encryptions of m to index i. Decoder-based versions of the other properties are defined similarly.

Now that we have decoder-based PLBE definitions that align with the decoder in the TT definitions, it is fairly straightforward to prove that the BSW transformation implies TT. The downside of introducing decoder-based PLBE definitions is that they are more difficult to work with as a target for

a construction. We address this issue by circling back to the original (BSW) PLBE definitions and augmenting them by allowing an attacker to make an a priori bounded number of queries to an encryption oracle. We show that 1-query PLBE implies decoder-based PLBE. This gives us an easier target (that is, 1-query PLBE).

Before describing the transformation from 1-query PLBE to decoder-based PLBE, we would like to point out that if the BSW definitions were augmented to allow an unbounded number of ciphertext queries, then decoder-based security follows immediately. For instance, let us consider the index hiding game. The reduction algorithm (that reduces unbounded-query PLBE to decoder-based PLBE) receives a decoder box D from the attacker. Given the unbounded queries, the reduction algorithm can measure (with reasonable accuracy) the success probabilities of D for indices $i-1$ and i, and therefore whether it can use D to distinguish between an encryption to index $i-1$ and i. However, with only 1 encryption query no such precise measurement is possible. Therefore, showing an attacker on decoder-based PLBE security implies an attacker on 1-query PLBE is a bit tricky. The reduction algorithm, after receiving the decoder box and message m from the adversary, chooses a random index $i^* \in \{i-1, i\}$, and queries the challenger for encryption of m for index i^*. It receives a ciphertext ct. Next, it queries the challenger with challenge message m and receives a challenge ciphertext ct^*. The reduction algorithm checks if $D(\mathsf{ct}) = D(\mathsf{ct}^*)$; if so, it guesses that m was encrypted for index i^*. We would like to point out that choosing query index i^* uniformly at

random from $\{i-1, i\}$ (as opposed to just fixing one of the two) is important for our analysis. The idea of running the decoder twice is sometimes referred to as the double-run trick [10, 32, 59]. The complete details of our analysis can be found in Section 5.3.

Impact on prior TT works using the PLBE framework. Traitor tracing schemes that had secret key tracing would need a new proof under the new PLBE definitions with 1-query allowed. We believe the bilinear map constructions [25, 70, 67] are likely secure under this definition, but showing this is outside scope of this paper. Note that the same problem is not present in PLBE with public trace-encryption (e.g., [27]), since the public key allows the reduction algorithm to generate ciphertexts.

2.1.2 Part 2: Constructing PLBE from Mixed FE

The hardness of constructing a PLBE scheme stems from the fact that it needs to satisfy the following two properties at the same time. First, a PLBE scheme needs to provide a predicate encryption (PE) like functionality where each secret key is associated with an "index," and each ciphertext is associated with an *index comparison* predicate. Also, the ciphertexts must not reveal any more information about the associated index comparison predicate other than what can be learned by running decryption. Second, the scheme must provide a broadcast encryption (BE) like compactness guarantee, which is that the size of ciphertexts must be short. In particular, the index needs to be represented in binary, which means the index comparison must be "sufficiently complex."

In this work, instead of directly building a PLBE scheme, we further reduce the task to constructing a new form of functional encryption scheme called Mixed FE. We show how Mixed FE can be combined with ABE for circuits to obtain PLBE. At a very high level, our approach is to decouple the functionality (delivering the message to users) and security requirements of a PLBE scheme, and to deal with them separately.

We begin by informally introducing the notion of Mixed FE. A Mixed FE scheme consists of setup, normal (or public key) encryption, *secret key encryption*, key generation, and decryption algorithms. The setup algorithm takes as input the security parameter λ and description of a function class $\mathcal{F}$ and outputs the public parameters pp and the master secret key msk. The normal encryption algorithm only takes as input the public parameters pp and outputs a (normal) ciphertext ct. The secret key encryption algorithm takes as input the master secret key msk and a function $f \in \mathcal{F}$ and outputs a (secret key) ciphertext ct. The key generation algorithm takes as input the master secret key msk and a message m and outputs a key sk_m. The decryption algorithm takes as input a ciphertext ct and a secret key sk_m and outputs a single bit. Now for correctness we require that decrypting a secret key encryption of any function f using a secret key sk_m outputs the evaluation of function f on message m, i.e., $f(m)$, whereas the decryption algorithm (almost) always outputs 1 when given a normal ciphertext as input, irrespective of the secret key used. Thus, one could visualize the normal encryption algorithm as always encrypting a "canonical" *always-accepting* function.

Intuitively, security states that no attacker should be able to distinguish between two ciphertexts that decrypt to the same values under all the secret keys in the attacker's possession. Now since there are two separate encryption algorithms, we have two different security properties. The first property says that secret key encryptions of two functions f_0 and f_1 should be indistinguishable if for every key in the attacker's possession the output of f_0, f_1 is identical. We call this function the indistinguishability property. The second property says that it should be hard to distinguish between a normal (public key) encryption and secret key encryption of a function f, where $f(m)$ must be equal to 1 for all keys sk_m in the attacker's possession. We call this the accept indistinguishability property.

We show that we can construct a PLBE scheme from a (key-policy) ABE scheme and a Mixed FE scheme. The idea is to encrypt a message using the ABE system with the attribute being set to be a Mixed FE ciphertext. Each user's secret key will be an ABE private key. Here the ABE private key is generated for the Mixed FE decryption circuit in which a Mixed FE secret key, corresponding to the user's index, is hardwired. The high level intuition is that when the attribute is a normal functional encryption ciphertext then all Mixed FE keys decrypt it to 1; thus any user with an appropriate ABE key could perform the decryption, whereas if the attribute is set to be a secret key ciphertext, then we can control the users who can decrypt it.

Formally, the scheme works as follows. During setup, the algorithm samples both ABE and Mixed FE key pairs $(\mathsf{abe.pp}, \mathsf{abe.msk})$, $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$.

To compute the ith user's private key, it samples a Mixed FE secret key $\mathsf{mixed.sk}_i$ for input i and also computes an ABE key $\mathsf{abe.sk}_i$ for predicate $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, i.e., Mixed FE decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired. And the ABE key $\mathsf{abe.sk}_i$ is set to be the ith user's private key. Now to encrypt a message m, the algorithm simply runs the ABE encryption algorithm with attributes set to be a Mixed FE ciphertext $\mathsf{ct}_{\mathsf{attr}}$. For standard PLBE encryption, $\mathsf{ct}_{\mathsf{attr}}$ is computed as a Mixed FE normal ciphertext, and for PLBE index encryption to some index i, $\mathsf{ct}_{\mathsf{attr}}$ is computed as a Mixed FE secret key encryption of function *greater than* i. Lastly, the PLBE decryption is the same as the ABE decryption algorithm.

Correctness can be observed directly. For standard PLBE ciphertext, $\mathsf{ct}_{\mathsf{attr}}$ is a normal FE ciphertext which decrypts to 1; thus the predicate, which is the decryption circuit $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, is satisfied for all i. Therefore, by ABE correctness, the ABE decryption algorithm will output the message m. For the PLBE index i ciphertext, $\mathsf{ct}_{\mathsf{attr}}$ is a Mixed FE secret key encryption of function "$> i$" which decrypts to 1 for all keys $\mathsf{mixed.sk}_j$ with $j > i$; thus the predicate is satisfied for all users with indices larger than i. Therefore, by ABE correctness, the ABE decryption algorithm will output the message m whenever $j > i$. For proving security, we rely on the fact that Mixed FE ciphertexts are indistinguishable to any adversary that does not have distinguishing secret keys. For instance, suppose there exists an adversary that can distinguish between PLBE normal encryptions and index 0 encryptions; then such an adversary can also be used to distinguish between Mixed FE normal ci-

phertexts and secret key ciphertexts encrypting function "> 0" (note that this is an always-accepting function). Thus, such an attack can be used to break the accept indistinguishability property of the Mixed FE scheme. Similarly, we can prove index hiding and message hiding security of the construction by reducing to Mixed FE and ABE (selective) security, respectively. Now if the Mixed FE scheme is 1-query secure, then so is the PLBE scheme.

Now the size of ciphertexts has only poly-log dependence on the number of users n as required. Because each user can be uniquely identified using a bit string of length $\log n$, so the length of the attribute (Mixed FE ciphertext) will be polynomial in $\log n$, and thus the PLBE ciphertext, which is in turn an ABE ciphertext, will have length polynomial in $\log n$ as well. Also, note that to use the above transformation it is sufficient to construct a Mixed FE scheme that supports comparison operation on $\log n$ bit strings. In this work, we show how to construct a Mixed FE scheme for any class of polynomial-sized branching program from the LWE assumption.[2] Our construction relies only on the polynomial hardness of LWE, although we require a superpolynomial modulus-to-noise ratio. Since we already have circuit ABE schemes from the LWE assumption [76, 22], combining that with our Mixed FE construction gives us collusion resistant traitor tracing from the LWE assumption as well.

Looking back, it is easy to observe that Mixed FE for branching programs that support comparison functionality is sufficient for our application.

[2]Note that this also gives us an alternate construction for selectively secure private key functional encryption with bounded collusions [110, 75].

However, as a design choice, here we instead chose to construct Mixed FE for general polynomial length branching programs as it is possible that this generalization leads to more applications in the future. Moreover, focusing on logarithmic length branching programs supporting comparisons, instead of general branching programs, did not lead to any significant simplification in the Mixed FE construction or its proof.

2.1.3 Part 3: An Enhanced LWE Toolkit

Before describing our LWE-based construction for Mixed FE, we define new "enhanced" properties for lattice trapdoors that will be useful in our work, and we believe it will find more applications in the future. In many LWE-based works, in addition to the LWE assumption itself, a critical tool has been the notion of lattice trapdoors [6, 72]. Lattice trapdoor samplers consist of a pair of algorithms, $\mathsf{TrapGen}$ and $\mathsf{SamplePre}$. The trapdoor generation algorithm $\mathsf{TrapGen}$ outputs a matrix $\mathbf{A}$ (that defines the lattice) and a trapdoor $T_{\mathbf{A}}$. The preimage sampling algorithm $\mathsf{SamplePre}$ takes as input a matrix $\mathbf{Z}$, a trapdoor for matrix $\mathbf{A}$, and a Gaussian parameter σ and outputs a matrix $\mathbf{U}$ such that $\mathbf{U}$ maps $\mathbf{A}$ to $\mathbf{Z}$ (that is, $\mathbf{A} \cdot \mathbf{U} = \mathbf{Z}$).[3]

These algorithms satisfy the following properties. First, the matrix $\mathbf{A}$ output by the trapdoor generation algorithm "looks like" a uniformly random matrix; we call this the *well-sampledness of matrix* property. Second, the

[3] Although the notion of preimage sampling is usually defined with respect to (w.r.t.) vectors instead of matrices, here we stick to using matrices for technical reasons discussed later in Chapter 6.

matrix output by SamplePre is indistinguishable from a matrix drawn from a discrete Gaussian distribution with parameter σ over the set of all matrices $\mathbf{V}$ such that $\mathbf{A} \cdot \mathbf{V} = \mathbf{Z}$. In particular, if $\mathbf{Z}$ is chosen uniformly at random, then the output of SamplePre "looks like" a matrix $\mathbf{U}$ drawn from a discrete Gaussian distribution with parameter σ; we call this the *preimage sampling property*. Lattice trapdoors with these properties have found a remarkable number of applications in building LWE-based cryptography.

In this work, we introduce two new *enhanced* properties for lattice trapdoors. The first property is the *row removal* property, which can be intuitively described as follows. Consider a setting where an adversary specifies some "target vectors," and the challenger must output a matrix $\mathbf{A}$ and a matrix $\mathbf{U}$ such that $\mathbf{U}$ maps some of the rows of $\mathbf{A}$ to the target vectors, and maps the remaining rows to uniformly random vectors. Then these rows targeting uniformly random vectors can be removed from the trapdoor sampling. In particular, the challenger can sample a shorter matrix $\mathbf{B}$ with a trapdoor, extend $\mathbf{B}$ with uniformly random vectors to get $\mathbf{A}$, and set $\mathbf{U}$ to be a matrix that maps $\mathbf{B}$ to the target vectors. These two scenarios will be indistinguishable for the PPT adversary.

The second property is called the *target switching* property. In this setting, consider an adversary that specifies two matrices, $\mathbf{Z}_0, \mathbf{Z}_1$, and a set of "target" indices such that the rows of $\mathbf{Z}_0$ and $\mathbf{Z}_1$ agree on these target indices. The challenger is supposed to sample a matrix $\mathbf{A}$ with a trapdoor,

compute a matrix $\mathbf{U}$ that maps $\mathbf{A}$ to $\mathbf{Z}_0$,[4] and output $\mathbf{U}$ together with the rows of $\mathbf{A}$ corresponding to the target indices *and only those rows*. Then the challenger can switch the $\mathbf{U}$ to map $\mathbf{A}$ to $\mathbf{Z}_1$, and the target switching property requires that this change is indistinguishable to the adversary (note that this would not be possible if the adversary receives any of the nontarget rows of $\mathbf{A}$). Moreover, the adversary is allowed to adaptively query for different target vectors/indices in both these games.

Now that we have these enhanced properties, let us discuss how to construct lattice trapdoors with these enhanced properties (using standard lattice trapdoors). Our construction is similar to the SampleLeft/SampleRight algorithms of [3, 40]. The enhanced trapdoor generation algorithm uses the standard trapdoor sampling algorithm to sample two matrices, $\mathbf{A}_1, \mathbf{A}_2$, together with the respective trapdoors $T_{\mathbf{A}_1}, T_{\mathbf{A}_2}$. It outputs $\mathbf{A} = [\mathbf{A}_1|\mathbf{A}_2]$ as the matrix and $T_{\mathbf{A}} = (T_{\mathbf{A}_1}, T_{\mathbf{A}_2})$ as the trapdoor. To sample a matrix $\mathbf{U}$ that maps $\mathbf{A}$ to $\mathbf{Z}$, the preimage sampling algorithm first chooses a uniformly random matrix $\mathbf{W}$ (of the same dimensions as $\mathbf{Z}$). It then uses $T_{\mathbf{A}_1}$ to compute a matrix $\mathbf{U}_1$ that maps $\mathbf{A}_1$ to $\mathbf{W}$, and uses $T_{\mathbf{A}_2}$ to compute a matrix $\mathbf{U}_2$ that maps $\mathbf{A}_2$ to $\mathbf{Z} - \mathbf{W}$. The final preimage matrix is set to be $\left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$. We use the matrix well-sampledness and preimage sampling properties of the standard lattice trapdoors to prove these enhanced properties; the detailed proof can be found in Section 6.2.

[4] Strictly speaking, we require $\mathbf{U}$ to map the target vectors of $\mathbf{A}$ to the target vectors of $\mathbf{Z}_0$, but the remaining vectors of $\mathbf{A}$ *approximately* map to the corresponding vectors of $\mathbf{Z}_0$.

2.1.4 Part 4: Constructing Mixed FE from LWE

Here we outline our Mixed FE construction for polynomial-sized (leveled) branching programs from the LWE assumption. The main ingredient of our construction is the "enhanced" lattice trapdoor sampling procedure $\mathsf{LT}_{\mathsf{en}} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ discussed above.

First, let us recall the notion of leveled branching programs. A leveled branching program of length ℓ and width w can be represented using w states per level, 2ℓ state transition functions $\pi_{j,b}$ for each level $j \leq \ell$, an input-selector function $\mathsf{inp}(\cdot)$ which determines the input read at each level, and an accepting and rejecting state. The program execution starts at state $\mathsf{st} = 1$ of level 1. Suppose the branching program reads the first input bit (say, b) at level 1 (i.e., $\mathsf{inp}(1) = 1$). Then the state of the program changes from st to $\pi_{1,b}(\mathsf{st})$. Such a process is carried out (iteratively) until the program's final state at level ℓ is computed. Depending upon the final state, the program either accepts or rejects.

For ease of exposition we will start with a simpler goal of constructing a 0-query secure Mixed FE scheme for a class of width-w read-once branching programs where each input bit is read once and in ascending order. Below we first outline a construction for such a 0-query system as it contains most of the central ideas, but is easier to digest. Later we discuss the modifications with which we can improve it to a secure 1-query scheme (and, more generally, q-query secure for any polynomial q) as well as expand the function class to arbitrary polynomial-sized branching programs.

Moving on to our 0-query Mixed FE construction, the master secret key consists of two sets of matrices and some trapdoor information. The first set, labeled as "randomization" matrices, consists of 4ℓ matrices $\{\mathbf{B}_{i,b}, \mathbf{C}_{i,b}\}_{i,b}$ for $i \in [\ell], b \in \{0,1\}$. The second set, labeled as "program" matrices, consists of $w\ell$ matrices $\{\mathbf{P}_{i,v}\}_{i,v}$ for $i \in [\ell], v \in [w]$. Here the $\mathbf{C}_{i,b}$ matrices are sampled uniformly at random from $\mathbb{Z}_q^{n\times m}$, whereas the remaining (randomization and program matrices) are sampled jointly with common trapdoors (per level). Basically, for each level $i \in [\ell]$, we sample a $(w+2)n \times m$ matrix $\mathbf{M}_i$ as

$$(\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{(w+2)n}, 1^m, q).$$

Now each $\mathbf{M}_i$ matrix is parsed as $w+2$ matrices of dimensions $n\times m$ stacked on top of each other, where the first two matrices are the randomization matrices and the remaining w matrices are the program matrices for the ith level. That is, for each i,

$$\begin{bmatrix} \mathbf{B}_{i,0} \\ \mathbf{B}_{i,1} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix} = \mathbf{M}_i.$$

All ℓ trapdoors $T_1, \ldots, T_\ell$ are stored as the trapdoor information in the master secret key. The public parameters, on the other hand, only include the matrix dimensions, LWE modulus, and noise parameters, but none of these matrices or trapdoor information.

At a high level, the encryption and key generation algorithms will adhere to the following structure. To (secret key) encrypt a branching program,

the trapdoors will be used to sample 2ℓ low norm matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ (two per level) such that each matrix $\mathbf{U}_{i,b}$ encodes the corresponding state transition function by mapping/targeting level i "program" matrices to level $i+1$ "program" matrices per the transition function $\pi_{i,b}$. Now the secret key for an input x will consist of $\ell+1$ key vectors $\{\mathbf{t}_i\}_i$. The first key component, $\mathbf{t}_1$, will contain the program matrix $\mathbf{P}_{1,1}$ (which represents the starting state) plus some randomization component generated using the level 1 randomization matrix $\mathbf{B}_{1,b}$. The remaining ℓ key vectors will have two components—the first component will cancel the previous randomization component, and the second component will add new randomization terms.[5] The idea is that if decryption is performed honestly, then all the randomization terms will get canceled and the final output will reflect the output of the branching program.

So this way the program matrices will be tied in such a manner that they encode the state transition information, and they can be used to perform the branching program execution. And the randomization matrices are added to make sure that (1) the computation is hidden at each step, and (2) if ciphertext matrices and key vectors are combined in any inadmissible way, then the randomization components do not get canceled. Let us now look at how to execute the above ideas.

Key generation. The key generation algorithm takes as input a string x and generates key vectors $\{\mathbf{t}_i\}_i$ as follows. It chooses ℓ uniform secret vectors

[5] Technically, the last key vector will only remove the previous randomization component. It doesn't add a new randomization term.

$\mathbf{s}_i \in \mathbb{Z}_q^n$ for $i \in [\ell]$ and $\ell+1$ noise vectors $\mathbf{e}_i \in \mathbb{Z}_q^m$ for $i \in [\ell+1]$. It also chooses a *short* secret vector $\widetilde{\mathbf{s}} \in \mathbb{Z}_q^n$ and sets key vectors as follows:

$$\forall\ i \in [\ell+1], \quad \mathbf{t}_i = \begin{cases} \mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{e}_1 & \text{if } i = 1, \\ -\mathbf{s}_{i-1} \cdot \mathbf{C}_{i-1,x_{i-1}} + \mathbf{s}_i \cdot \mathbf{B}_{i,x_i} + \mathbf{e}_i & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell \cdot \mathbf{C}_{\ell,x_\ell} + \mathbf{e}_{\ell+1} & \text{if } i = \ell+1. \end{cases}$$

In words, the randomization component (likewise, cancellation component) added in the ith key vector ($(i+1)$th key vector) is an LWE sample where the public matrix used depends on the ith bit of input x. Looking ahead, choosing the "randomization" matrices depending on the string x would assert that the ciphertext matrices cannot be arbitrarily combined to learn meaningful terms.

Normal encryption. The normal (public key) encryption algorithm simply samples 2ℓ random short matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ as $\mathbf{U}_{i,b} \leftarrow \chi^{m \times m}$, where χ is the noise distribution chosen during setup.

Secret key encryption. Moving on to the secret key encryption algorithm, on input the master secret key and a branching program $\mathsf{BP} = \left(\{\pi_{i,b}\}_{i,b}, \mathsf{acc}, \mathsf{rej}\right)$, it samples low norm matrices $\{\mathbf{U}_{i,b}\}$ as follows. It first chooses two "program" matrices for the last level $\ell+1$ as $\mathbf{P}_{\ell+1,\mathsf{rej}} = \mathbf{0}^{n \times m}$ and $\mathbf{P}_{\ell+1,\mathsf{acc}} \leftarrow \mathbb{Z}_q^{n \times m}$. That is, for the accepting state, it chooses a random program matrix, and for the rejecting state it sets the matrix to be all zeros. Next, using the ith trapdoor T_i (included in the master secret key) it runs the $\mathsf{EnSamplePre}$ algorithm to sample the ciphertext (transition) matrices

$\mathbf{U}_{i,0}, \mathbf{U}_{i,1}$ such that they map/target matrix $\mathbf{M}_i$ as follows:

$$\begin{bmatrix} \mathbf{B}_{i,0} \\ \mathbf{B}_{i,1} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix} \xrightarrow{\mathbf{U}_{i,0}} \begin{bmatrix} \mathbf{C}_{i,0} \\ \$ \\ \mathbf{P}_{i+1,\pi_{i,0}(1)} \\ \vdots \\ \mathbf{P}_{i+1,\pi_{i,0}(w)} \end{bmatrix}, \qquad \begin{bmatrix} \mathbf{B}_{i,0} \\ \mathbf{B}_{i,1} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix} \xrightarrow{\mathbf{U}_{i,1}} \begin{bmatrix} \$ \\ \mathbf{C}_{i,1} \\ \mathbf{P}_{i+1,\pi_{i,1}(1)} \\ \vdots \\ \mathbf{P}_{i+1,\pi_{i,1}(w)} \end{bmatrix}.$$

Here we use "$" to denote a uniformly random $n \times m$ matrix of appropriate dimension. In words, the structure we enforce here is that the matrix $\mathbf{U}_{i,b}$ targets the $\mathbf{B}_{i,b}$ randomization matrix to its $\mathbf{C}_{i,b}$ counterpart, and the $\mathbf{B}_{i,1-b}$ randomization matrix to a random matrix. Additionally, $\mathbf{U}_{i,b}$ encodes the information about transition function $\pi_{i,b}$ by targeting the level i program matrices to their level $i+1$ counterparts per $\pi_{i,b}$. Thus, from the perspective of both correctness and security, this guarantees that a key vector $\mathbf{t}_i$ for some input x must be combined with ciphertext component $\mathbf{U}_{i,x_i}$, as otherwise the randomization matrix would be mapped to a random matrix, thereby destroying the underlying structure.

Decryption. First, let us focus on decrypting a secret key encryption of branching program BP using a secret key $\{\mathbf{t}_i\}_i$ corresponding to an input x. Intuitively, one could visualize the matrices $\{\mathbf{U}_{i,0}, \mathbf{U}_{i,1}\}_i$ in the ciphertext as "encodings" of the branching program state transition functions $\pi_{i,0}, \pi_{i,1}$, respectively. Therefore, decrypting the ciphertext using a secret key for some input x will be analogous to evaluating the branching programs BP on input x directly. Recall that we assumed (for ease of exposition) that the branching programs are read-once, and input bits are read sequentially in ascending

order. Thus, the first input bit x_1 is read at level 1. Then evaluation of BP at level 1 would map the state $\mathsf{st}_1 = 1$ at level 1 to state $\mathsf{st}_2 = \pi_{1,x_1}(1)$ at level 2. Analogously, the decryptor can compute

$$\begin{aligned}
\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \mathbf{t}_2 &\approx (\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}) \cdot \mathbf{U}_{1,x_1} + \mathbf{t}_2 \\
&\approx \mathbf{s}_1 \cdot \mathbf{C}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{t}_2 \\
&\approx \cancel{\mathbf{s}_1 \cdot \mathbf{C}_{1,x_1}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + (\cancel{-\mathbf{s}_1 \cdot \mathbf{C}_{1,x_1}} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2}) \\
&\approx \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2}.
\end{aligned}$$

In general, if the program state at level i during execution is st_i, then the decryptor will accumulate the term of the form $\mathbf{s}_i \cdot \mathbf{B}_{i,x_i} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i,\mathsf{st}_i}$ by successively summing and multiplying secret key and ciphertext components as

$$(\cdots((\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \mathbf{t}_2) \cdot \mathbf{U}_{2,x_2} + \mathbf{t}_3) \cdots + \mathbf{t}_i)\,.$$

This can be verified as follows. We know that the bit read at level i is x_i, and thus the new state at level $i+1$ will be $\mathsf{st}_{i+1} = \pi_{i,x_i}(\mathsf{st}_i)$. Now the accumulated sum-product during decryption will be

$$\begin{aligned}
&(\mathbf{s}_i \cdot \mathbf{B}_{i,x_i} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i,\mathsf{st}_i}) \cdot \mathbf{U}_{i,x_i} + \mathbf{t}_{i+1} \\
&\qquad\approx \cancel{\mathbf{s}_i \cdot \mathbf{C}_{i,x_i}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i+1,\mathsf{st}_{i+1}} + \left(\cancel{-\mathbf{s}_i \cdot \mathbf{C}_{i,x_i}} + \mathbf{s}_{i+1} \cdot \mathbf{B}_{i+1,x_{i+1}}\right) \\
&\qquad\approx \mathbf{s}_{i+1} \cdot \mathbf{B}_{i+1,x_{i+1}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i+1,\mathsf{st}_{i+1}}.
\end{aligned}$$

Therefore, the invariant is maintained. Continuing in this way, the decryptor can iteratively compute the sum-product combining all key and ciphertext components. Note that (by definition) adding in the $(\ell+1)$th key component

t_ℓ does not introduce a term like $\mathbf{s}_{\ell+1} \cdot \mathbf{B}_{\ell+1,x_{\ell+1}}$ to the sum-product; thus the accumulated term at the top will be $\approx \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}}$, where $\mathsf{st}_{\ell+1}$ is either acc or rej depending on $\mathsf{BP}(x)$. Finally, the decryptor simply checks whether the norm of the final sum-product term is small or not. Recall that the program matrix for the last level corresponding to the rejecting state is set to be all zeros, i.e., $\mathbf{P}_{\ell+1,\mathsf{rej}} = \mathbf{0}^{n\times m}$. Therefore, if $\mathsf{BP}(x) = 0$, then the norm of the final sum-product term will be small, which the decryptor can test and output 0. Otherwise, with high probability the final sum-product term will be large and it outputs 1.

By the above analysis, correctness follows in the case where ciphertext is a secret key encryption. The correctness of decryption when the ciphertext is a normal (public key) ciphertext follows from the fact that the ciphertext matrices $\{\mathbf{U}_{i,0}, \mathbf{U}_{i,1}\}_i$ are independently sampled random short matrices.

0-*query security.* To prove 0-query security of our construction, we need to argue that it satisfies both function indistinguishability as well as accept indistinguishability properties. We start by proving the function indistinguishability security. Recall that in the 0-query function indistinguishability security game, an adversary submits two branching programs, $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$, and is allowed to make a polynomial number of key queries such that for each queried input x, $\mathsf{BP}^{(0)}(x) = \mathsf{BP}^{(1)}(x)$ (i.e., every secret key given out has same output on both the challenge programs). The adversary receives secret key encryption of either $\mathsf{BP}^{(0)}$ or $\mathsf{BP}^{(1)}$, and its goal is to distinguish between them.

Although the full security proof is technically involved, the main ideas

behind our proof are very intuitive. Before diving into the proof structure, we point out that the construction described above has to be slightly modified for proving security. Below we describe our proof ideas as well as discuss the modifications required along the way.

At a high level, our idea is to "hardwire" the output of the challenge branching programs in every secret key given to the adversary. Note that the security definition states that both challenge programs must evaluate to the same value on all queried inputs, and thus we only need to hardwire a single value in each key. For ease of exposition, assume that the adversary makes exactly one secret key query. (In the general case of polynomially many key queries, the proof proceeds by hardwiring the level 1 components in all secret keys, followed by level 2 hardwiring, and so on.) Let $\{\mathbf{U}_{i,b}\}_{i,b}$ be the challenge ciphertext, and let $\{\mathbf{t}_i\}_i$ be the secret key computed by the challenger. Our hardwiring strategy works as follows. We start by rewriting the second secret key vector $\mathbf{t}_2$ in terms of $\mathbf{t}_1$ as follows:

$$\begin{aligned}
\mathbf{t}_2 &= -\mathbf{s}_1 \cdot \mathbf{C}_{1,x_1} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} \cdot \mathbf{U}_{1,x_1} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} \cdot \mathbf{U}_{1,x_1} - \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -\left(\mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}\right) \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \\
&= -(\mathbf{t}_1 - \mathbf{e}_1) \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2.
\end{aligned}$$

Here st_2 is the state of the challenge branching program encrypted (after one step is executed). Now in the above term, we can *smudge* the term $\mathbf{e}_1 \cdot \mathbf{U}_{1,x_1}$

by appropriately choosing the noise distributions, i.e., $\mathbf{e}_2 \gg \mathbf{e}_1 \cdot \mathbf{U}_{1,x_1}$.[6] (Note that since we require smudging here, thus the LWE modulus q needs to be superpolynomial in the lattice dimension.) Thus, the second key component can be indistinguishably computed as follows without requiring any explicit knowledge of the $\mathbf{C}_{1,x_1}$ matrix:

$$\mathbf{t}_1 = \mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{e}_1,$$

$$\mathbf{t}_2 = \begin{array}{c} -(\mathbf{t}_1 - \mathbf{e}_1) \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} \\ + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2 \end{array} \xrightarrow{\text{Smudging}} \mathbf{t}_2 = \begin{array}{c} -\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} \\ + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2. \end{array}$$

Next, we use the row removal property of our enhanced trapdoor sampling algorithms to remove the $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ rows from the first matrix $\mathbf{M}_1$ and instead sample these randomly. To understand why this can be done, recall that in the actual construction the encryptor needs the ability to create a ciphertext for *any* branching program that could be chosen even after all the keys have been distributed. That is, the encryptor must be able to sample matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ such that they map level 1 program matrices $\{\mathbf{P}_{1,v}\}_v$ to level 2 program matrices $\{\mathbf{P}_{2,v}\}_v$ per some transition functions $\{\pi_{1,b}\}_b$, as well as ensure that $\mathbf{B}_{1,b} \cdot \mathbf{U}_{1,b} = \mathbf{C}_{1,b}$. Now since the keys contain the matrices $\mathbf{C}_{1,0}, \mathbf{C}_{1,1}$ and they could be given out even before matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ are sampled, thus matrices $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ must be sampled together with $\{\mathbf{P}_{1,v}\}_v$ such that they share a common trapdoor.

[6] If we keep on smudging this way, our noise distributions will have to grow by an exponential factor at each step. In the main body, we show how to avoid this by a better smudging argument.

However, at this stage in the proof the challenge branching program is (selectively) fixed ahead of any secret key queries. Therefore, in this context we can sample matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ to only map level 1 program matrices to their level 2 counterparts, and simply set the matrices $\mathbf{C}_{1,b}$ as $\mathbf{C}_{1,b} = \mathbf{B}_{1,b} \cdot \mathbf{U}_{1,b}$ and use these to compute the secret keys. We would like to point out that in order to perform this row removal securely, it is important that $\mathbf{B}_{1,b} \cdot \mathbf{U}_{1,1-b} = \$$, that is, matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ map both matrices $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ to random and uncorrelated matrices.

Now once we have removed the $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ rows from the first matrix, we use the LWE assumption to switch the first key component $\mathbf{t}_1$ to a random vector. Note that at this point since matrices $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ are sampled uniformly (i.e., are no longer sampled with trapdoor information) and secret vector $\mathbf{s}_1$ is not used in computing the second key component $\mathbf{t}_2$, thus we can apply LWE to switch $\mathbf{t}_1$ to random, where the LWE secret is $\mathbf{s}_1$ and the LWE public matrix will be $\mathbf{B}_{1,x_1}$.[7] Concretely, using LWE we can perform the following switch, which essentially erases the information about the level 1 program matrix $\mathbf{P}_{1,1}$ from the secret keys, thereby rendering the program evaluation to start from level 2 and state st_2 instead:

$$\mathbf{t}_1 = \mathbf{s}_1 \cdot \mathbf{B}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{e}_1 \quad \xrightarrow{\mathsf{LWE}} \quad \mathbf{t}_1 = \$,$$

$$\mathbf{t}_2 = -\mathbf{t}_1 \cdot \mathbf{U}_{1,x_1} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{2,\mathsf{st}_2} + \mathbf{s}_2 \cdot \mathbf{B}_{2,x_2} + \mathbf{e}_2.$$

[7] In the general case of multiple key queries the LWE public matrices will be both $\mathbf{B}_{1,0}, \mathbf{B}_{1,1}$ and the LWE secret will consist of all the secret key vectors $\mathbf{s}_i$ that are chosen independently and *per key*.

Now iteratively performing this *hardwiring* strategy (ℓ times), we end up switching all but the last key components to be random vectors. Also, the last key component will contain the final program matrix, which is either a random matrix or a zero matrix, depending on the program output. Thus, the key vectors contain no information about the "program" matrices chosen during setup. At this point, the challenge matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ still contain the information about the branching program encrypted in the form of mapping between level i and $i+1$ "program" matrices, i.e., the state transition functions $\{\pi_{i,b}\}_{i,b}$. Finally, to argue indistinguishability here (i.e., between the challenge matrices) we use the target switching property of our enhanced trapdoors. We apply a bottom-up approach to execute this change. First, note that the level 1 program matrices do not explicitly appear anywhere, except that they are used to sample the level 1 ciphertext matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$. Thus, we can use the target switching property to switch the targets of matrices $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$. Observe that this now removes the information about level 2 program matrices as well as the level 1 transition functions of challenge branching programs. Next, by the same principle, we can perform the same target switching step for $\mathbf{U}_{2,0}, \mathbf{U}_{2,1}$ and continue so on. If we keep on performing the *target switching* step this way until the top, then the challenge ciphertext will contain no information about the challenge programs (i.e., their state transition functions), thereby completing our claim of function indistinguishability.[8] This completes the first

[8]Technically, we cannot apply the target switching property here, because the target switching property only guarantees that targets being switched are approximately mapped, whereas here we target exactly. Therefore, we also need to add some noise in the targeted

proof.

The proof of the 0-query accept indistinguishability security of our construction is similar, but more technical due to the fact that we need to argue that the challenge ciphertext is indistinguishable from random short matrices.

However, for our PLBE construction, the Mixed FE scheme must handle one ciphertext query as well, and it is not clear how to prove the above construction to be 1-query secure directly. The bottleneck is the fact that in the above proof strategy we *hardwire* all secret key components to match the output of the challenge program. Now if the adversary is allowed to make a secret key encryption query, then it is not clear how the challenger would still program the secret key vectors. To get around this problem, we expand our system such that it consists of λ *pairs* of 0-query subsystems. Very briefly, during encryption, the algorithm now also samples a λ-bit string tag randomly and depending on each bit of tag, it chooses one subsystem in each pair and runs the 0-query encryption for that subsystem. Now during key generation, it (linearly) secret shares the starting program across these λ pairs of subsystems such that the same secret share is used for both subsystems in each pair. Then it runs the 0-query key generation algorithm for all these 2λ subsystems with their corresponding secret shares as the starting program matrices. For decryption, the subsystems chosen during encryption are combined with their counterparts in the secret keys, and 0-query decryption is performed in these

"program" matrices before running $\mathsf{EnSamplePre}$ algorithm. For simplicity, we avoid this modification.

subsystems along with a (linear) reconstruction on top of the output. More details are provided in the main body.[9]

This completes the technical overview of our construction.

Relation to recent LWE-based schemes. There have been several recent works that have advanced the state of the art in computing branching programs on encrypted data with the goal of reducing security to LWE or LWE-like assumptions [71, 37, 15, 81, 39, 80, 117]. While our construction above benefits from that lineage, we wish to briefly call out a few important distinctions.

First, from a purely mechanical perspective, the construction of our Mixed FE scheme is structurally very different from the constructions of the aforementioned primitives. Very briefly, in all previous constructions the evaluator multiplies a set of matrices and sums them up to get the final output, whereas in our construction, we do not use this "one-shot" approach for evaluation. Instead, we multiply a component from the ciphertext with a secret key component, then add in another secret key component, multiply this sum with another ciphertext component, and so on. Thus, our mechanism of combining the secret key and ciphertext components is much different than what was used in prior works.[10]

[9] We would like to point out that the above idea could also be used to improve the Mixed FE construction to be q-query secure for any polynomial q. The idea will be to sample tag strings tag from a larger alphabet instead of $\{0,1\}$. However, we only focus on 1-query security, as it is sufficient for our result.

[10] Although one can always express such a nested matrix multiplication and addition mechanism using only a sequence of matrix multiplications with much larger (and repetitive) matrices, we point out that the underlying structure of such matrices as well as the modified

Second, we structure our proof of security to hardwire the outputs for keys one level at a time until we hit the final output level in which we have the final outputs hardwired but lost information about the program that got us there. In this sense at a high level this leveled programming proof structure much more closely resembles that of garbled circuit proofs. Thus, we need to develop new lower LWE-specific techniques to match these goals. In contrast, works such as [37, 81, 80, 117] have a different aim of losing all meaningful information when a secret is not known.

2.2 Embedding indentities in traitor tracing

We start by formally defining the notion of embedded identity traitor tracing (EITT) systems. In order to capture a broader class of traitor tracing systems, we consider three different variants for embedded identity tracing — (1) indexed EITT, (2) bounded EITT, and (3) full (unbounded) EITT. Although the notion of full/unbounded EITT is the most general notion we define and therefore it is also likely the most desirable notion, we believe that both indexed and bounded EITT systems will also find many direct applications as will be evident later during their descriptions. In addition, we also show direct connections between all three notions by providing different transformations among these notions.

Next, we move on to realizing these EITT systems under standard

evaluation algorithm will still be much different from those used in previous works.

assumptions. To that end, we first introduce a new intermediate primitive which we call *embedded-identity private linear broadcast encryption* (EIPLBE) that we eventually use to build EITT schemes. As the name suggests, the notion of EIPLBE is inspired by and is an extension of *private linear broadcast encryption* (PLBE) schemes. In this work, we show that the above-stated extension of PLBE systems can be very useful in that it leads to new solutions for the embedded identity traitor tracing problem.

Finally, we instantiate our EIPLBE scheme under the LWE assumption.

2.2.1 Embedded Identity Traitor Tracing Definitions

Let us first formally recall the notion of standard traitor tracing (i.e., without embedding identities in the secret keys). A traitor tracing system consists of four poly-time algorithms — Setup, Enc, Dec, and Trace. The setup algorithm takes as input security parameter λ, and number of users n and generates a public key pk, a tracing key key, and n private keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$. The encryption algorithm encrypts a message m using public key pk, and the decryption algorithm decrypts a ciphertext using any one of the private keys sk_i. The tracing algorithm takes tracing key key, two messages m_0, m_1 as input, and is given (black-box) oracle access to a pirate decoding algorithm D.[11] It outputs a set $S \subseteq [n]$ of users signalling that the keys sk_j for $j \in S$

[11] Traditionally, the tracing algorithm was defined to work only if the decoder box could decrypt encryptions of random messages. However, as discussed in [79], this definition does not capture many practical scenarios. Therefore we work with a broader abstraction where the trace algorithm works even if the decoder can only distinguish between encryptions of

were used to create the pirate decoder D. The security requirements are as described in the previous section.

Let us now look at how to embed identities in the private user keys such that the tracing algorithm outputs a set of identities instead. Below we describe the identity embedding abstractions considered in this work. Throughout this thesis, κ denotes the length of identities embedded (that is, identity space is $\{0,1\}^\kappa$).

Indexed EITT. We begin with indexed EITT as the simplest way to introduce identity embedding functionality in the standard TT framework is as follows. The setup algorithm takes both n and κ as inputs and outputs a master secret key msk. Such systems will have a special key generation algorithm that takes as input msk along with an index-identity pair $(i, \mathsf{id}) \in [n] \times \{0,1\}^\kappa$, and outputs a user key $\mathsf{sk}_{i,\mathsf{id}}$. When the i^{th} user requests a key then it can supply its identity id, and the authority runs key generation on corresponding inputs to sample a secret key for that particular user.

Encryption, decryption, and tracing algorithms remain unaffected with the exception that the tracing algorithm outputs a set of user identities $S \subseteq \{0,1\}^\kappa$ instead.[12] Now the IND-CPA and secure tracing requirements very naturally extend to indexed EITT systems with one caveat that the adversary

two specific messages.

[12] Although one could ask the tracer to output a set of index-identity pairs instead of only identities, this seems unnecessary as the user index can always be embedded in its identity.

can only obtain a user key for each index at most once in the traitor tracing game. Comparing this with standard TT schemes in which each corrupted user receives a unique private key depending on its index, this constraint on set of corruptible keys is a natural translation.

Looking carefully at the above abstraction, we observe that using such indexed systems in practice would seem to resolve the 'look-up table' problem thereby allowing third party tracing, but the problem of statefulness is not yet completely resolved. Concretely, the key generating authority still needs to maintain a counter (that is $\log(n)$ bits) which represents the number of keys issued until that point. Basically each time someone queries for a secret key for identity id, the authority generates a secret key for identity id and index being the current counter value, and it increments the counter in parallel. This constraint stems from the fact that for guaranteeing correct tracing it is essential that the adversary receives at most one key per index $i \in [n]$. Although for a lot of applications indexed EITT might already be sufficient, it is possible that for others this is still restrictive. To that end, we define another EITT notion to completely remove the state as follows.

Bounded EITT. The idea behind bounded EITT is that now the input n given to the setup algorithm represents an upper bound on the number of keys an adversary is allowed to corrupt while the system still guarantees correct traceability. And importantly, the key generation algorithm now only receives an identity id as input instead of an index-identity pair. Thus, the

authority does not need to maintain the counter, that is it does not need to keep track of number of users registered. Another point of emphasis is that in a Bounded EITT system if the number of keys an attacker corrupts exceeds the setup threshold n, the attacker may avoid being traced; however, even in this scenario tracing procedure will not falsely indict an non-colluding user. In addition to being a useful property in its own right, the non-false indictment property will be critical in amplifying to Unbounded EITT.

Interestingly, we show a generic transformation from any indexed EITT scheme to a bounded EITT scheme with only a minor efficiency loss. More details on this transformation are provided towards the end of this section. Looking ahead, this transformation only relies on the existence of signatures additionally.

Unbounded EITT. Lastly the most general notion of embedded identity traitor tracing possible is of systems in which the setup algorithm only takes κ the length of identities as input, thus there is no upper bound on the number of admissible corruptions set during setup time. Therefore, the adversary can possibly corrupt an arbitrary (but polynomial) number of users in the system. In this work, we additionally provide an efficient unconditional transformation from bounded EITT schemes to unbounded EITT schemes thereby completely solving the embedded identity tracing problem. More details on this transformation are also provided towards the end of this section.

Next, we move on to building the indexed EITT schemes under standard

assumptions. As discussed before, we first introduce the intermediate notion of EIPLBE.

2.2.2 Embedded-Identity Private Linear Broadcast Encryption

Let us start by recalling the notion of private linear broadcast encryption (PLBE) [25]. Syntactically, a PLBE scheme is same as a traitor tracing scheme as in it consists of setup, key generation, encryption, decryption algorithms with the exception that instead of tracing algorithm it provides an additional encryption algorithm usually referred to as *index-encryption* algorithm. In PLBE systems, the setup algorithm outputs a public, master secret, and index-encryption key tuple $(\mathsf{pk}, \mathsf{msk}, \mathsf{key})$. As in TT systems, the key generation uses master secret key to sample user private keys sk_j for any given index $j \in [n]$, while the PLBE encryption algorithm uses the public key to encrypt messages. The index-encryption algorithm on the other hand uses the index-encryption key to encrypt messages with respect to an index i. Now such a ciphertext can be decrypted using sk_j only if $j \geq i$, thus one could consider such ciphertexts as encrypting messages under the comparison predicate '$\geq i$'.

BSW showed that the PLBE framework could be very useful for building TT systems. An inconvenient limitation of their tracing schema is that during tracing the algorithm essentially runs a brute force search over set of user indices $\{1, 2, \ldots, n\}$ to look for traitors. This turns out to be problematic if we want to embed polynomial length identities in the secret keys. Because

now the search space for traitors is exponential which turns the above brute force search mechanism rather useless. Thus it is not very clear whether the PLBE framework is an accurate abstraction for 'embedded identity' TT.

In this work, our intuition is to extend the PLBE framework such that it becomes more conducive for implementing the embedded identity tracing functionality in TT systems. Hence, we propose a new PLBE framework called embedded-identity PLBE. As in PLBE, an EIPLBE scheme consists of a setup, key generation, encryption, decryption and special-encryption algorithm. (Here special-encryption algorithm is meant to replace/extend the index-encryption algorithm provided in general PLBE schemes.) Semantically, the differences between PLBE and EIPLBE are as follows. In EIPLBE, the user keys are associated with an index-identity pair (j, id). And, special-encryptions are associated with a index-position-bit tuple (i, ℓ, b), where position is a symbol in $[\kappa] \cup \{\perp\}$. The special-encryption ciphertexts can further be categorized into two types:

($\ell = \perp$) In this case the special-encryption ciphertext for index-position-bit tuple $(i, \ell = \perp, b)$ behaves identical to a PLBE index-encryption to index i. That is, such ciphertexts can be decrypted using $\mathsf{sk}_{j,\mathsf{id}}$ as long as $j \geq i$.

($\ell \neq \perp$) In this case the ciphertext can be decrypted using $\mathsf{sk}_{j,\mathsf{id}}$ as long as either $j \geq i+1$ or $(j, \mathsf{id}_\ell) = (i, 1-b)$. In words, these ciphertexts behave same as a PLBE index-encryption to index i, except decryption by the users corresponding to index-identity pair (i, id) is also disallowed if ℓ^{th}

bit of their id matches bit value b.

In short, the special-encryption algorithm (when compared with PLBE index-encryption) provides an additional capability of disabling decryption ability of users depending upon a single bit of their identity. The central idea behind introducing this new capability is that it facilitates a simple mechanism for tracing the identity bit-by-bit. The tracing algorithm runs as a two-step process where the first phase is exactly same as in the PLBE to TT transformation which is to trace the indices of corrupt users. This can be executed as before by using the PLBE functionality of disabling each index one-by-one, that is estimate successful decryption probability of encryptions to indices in 1 to $n+1$ while keeping position variable $\ell = \perp$. This is followed by the core *identity tracing phase* in which the tracing algorithm performs a sub-search on each user index i where it noticed a gap in first phase. Basically the sub-search corresponds to picking a target index obtained during first phase, and then sequentially testing whether the ℓ^{th} bit in the corrupted identity is zero or one for all positions $\ell \in [\kappa]$. And, this is where the above additional disabling capability is used.

Next we discuss the expanded set of security properties required from EIPLBE. More details on the above transformation are provided afterwards.

normal-hiding. Standard encryptions are indistinguishable from special-encryptions to $(1, \perp, 0)$.

index-hiding. Special-encryptions to $(i, \perp, 0)$ are indistinguishable from special-encryptions to $(i+1, \perp, 0)$ if an adversary has no secret key for index i.

lower-ID-hiding. Special-encryptions to $(i, \perp, 0)$ are indistinguishable from special-encryptions to (i, ℓ, b) if an adversary has no secret key for index i and identity id such that $\mathsf{id}_\ell = b$.

upper-ID-hiding. Special-encryptions to $(i+1, \perp, 0)$ are indistinguishable from special-encryptions to (i, ℓ, b) if an adversary has no secret key for index i and identity id such that $\mathsf{id}_\ell = 1 - b$.

message-hiding. Special-encryptions to $(n+1, \perp, 0)$ hide the message encrypted.

Building Indexed EITT from EIPLBE. The setup, key generation, encryption and decryption algorithms for the tracing scheme are same as that for the underlying EIPLBE scheme. Let us now look at how to trace identities from the pirate decoding device. As mentioned before, the tracing proceeds in two phases — (1) index tracing, followed by (2) identity tracing. The idea is to first trace the set of indices of the corrupted users, say $S_{\mathsf{indx}} \subseteq [n]$, and then in the second phase for each index $i \in S_{\mathsf{indx}}$, the tracer will (bit-by-bit) extract the corresponding identity corrupted. Formally, the tracing proceeds as follows

Phase 1. For $i \in [n+1]$, do the following:

A. Compute polynomially many special-encryptions to index-position-bit $(i, \perp, 0)$.

B. Run decoder D on each ciphertext individually to test whether it decrypts correctly or not. Let $\hat{p}_i$ denote the fraction of successful decryptions.

Let S_{indx} denote the set of indices i of such that $\hat{p}_i$ and $\hat{p}_{i+1}$ are noticeably far.

Phase 2. Next, for each $i \in S_{\mathsf{indx}}$ and $\ell \in [\kappa]$, do the following:

A. Compute polynomially many special-encryptions to index-position-bit $(i, \ell, 0)$.

B. Run decoder D on each ciphertext individually to test whether it decrypts correctly or not. Let $\hat{q}_{i,\ell}$ denote the fraction of successful decryptions.

Output Phase. Finally, for each $i \in S_{\mathsf{indx}}$, it sets the associated traced identity id as follows. For each $\ell \in [\kappa]$, if $\hat{p}_i$ and $\hat{q}_{i,\ell}$ are noticeably far, then set ℓ^{th} bit of id to be 0, else sets it to be 1.

Let us now see why this tracing algorithm works. In the above procedure, the first phase (index tracing) is identical to the PLBE-based tracing algorithm. Thus, by a similar argument it follows that if $i \in S_{\mathsf{indx}}$, then it suggests that the decoder D was created using a key corresponding to index-identity pair (i, id)

for some identity id. (This part of the argument only relies on normal-hiding, index-hiding and message-hiding security properties.)

The more interesting component of the tracing algorithm is the *identity tracing* phase (i.e., phase 2). The idea here is to selectively disable the decryption ability of users for a fixed index if a particular bit in their identities is 0. Recall that an adversary can not distinguish between special-encryptions to tuple $(i, \perp, 0)$ and $(i, \ell, 0)$ as long as it does not have any secret key for (i, id) such that $\mathsf{id}_\ell = 0$. This follows from 'lower-ID-hiding' property. Similarly, an adversary can not distinguish between special-encryptions to tuple $(i+1, \perp, 0)$ and $(i, \ell, 0)$ as long as it does not have any secret key for (i, id) such that $\mathsf{id}_\ell = 1$. This follows from 'upper-ID-hiding' property. Now whenever $i \in S_{\mathsf{indx}}$ we know that $\hat{p}_i$ and $\hat{p}_{i+1}$ are noticeably far. Also, recall that in indexed EITT tracing definition the adversary is allowed to key query for *at most* one identity per index. Therefore, the estimate $\hat{q}_{i,\ell}$ will either be close to $\hat{p}_i$ or to $\hat{p}_{i+1}$, as otherwise one of upper/lower-ID-hiding properties will be violated. Combining all these observations, we can prove correctness/security of the above tracing algorithm.

2.2.3 Building EIPLBE from LWE

Our EIPLBE scheme also follows the Mixed FE+ABE approach described in the previous section. Here instead of the comparison function, the Mixed FE ciphertexts in our scheme will be for more expressive functions. In particular, it suffices to have a Mixed FE scheme where the functions are pa-

rameterized by (y^*, ℓ^*, b^*), and it checks if input (y, id) either satisfies $y > y^*$, or $y = y^*$ and $\mathsf{id}_\ell \neq b^*$. Since such simple functions can be implemented in log-depth, we can use the ABE+Mixed FE approach for building EIPLBE as well.

2.2.4 Indexed Embedded-Identity TT to Bounded Embedded-Identity TT

In this part, we discuss our transformation from a tracing scheme with indexed key generation to one where there is no index involved, but the correct trace guarantee holds only if total number of keys is less than an apriori set bound. For technical reasons we require the bounded EITT system to provide a stronger false tracing guarantee, which states there should be no false trace even if the adversary obtains an unbounded (but polynomial) number of keys. Looking ahead, this property will be crucial for the transformation from bounded EITT to its unbounded counterpart.

The high-level idea is to have λ different strands, and in each strand, we have a separate indexed-system with a large enough index bound (that depends on the bound on number of keys n). When generating a key, we choose λ random indices (within the index bound) and generate λ different keys for the same identity in the different strands using the respective randomly chosen indices. Now, we will set the index bound to be n^2, and as a result, at least one strand has all distinct indices (with overwhelming probability). To (special-)encrypt a message, we secret-share the message in the λ different strands, and

encrypt them separately. This approach satisfies the correct-trace guarantee, but does not satisfy the false-trace guarantee. In particular, note that the false-trace guarantee should hold even if the number of key queries is more than the query bound. This means the underlying indexed scheme should not report a false trace even if there are multiple identities for a index, which is a strictly stronger false-trace guarantee for the underlying system (and our system does not satisfy it).

There is an elegant fix to this issue. Instead of generating keys for the queried identity id, the key generation algorithm now generates a signature on id, and generates keys for (id, σ). This fixes the false-trace issue. Even if an adversary queries for many secret keys, if it is able to produce a decoding box that can implicate a honest user, then that means this box is able to forge signatures, thereby breaking the signature scheme's security. We describe the scheme a little more formally now.

To build a tracing scheme with bound n, the setup algorithm chooses λ different public/secret/tracing keys for the indexed scheme with index bound set to be n^2. The setup algorithm also chooses a signature key/verification key. It sets the λ different public keys and the verification key to be the new public key, and similarly the master secret key has the λ different master secret keys and the signature key. Encryption of a message m works as follows: the encryption algorithm chooses λ shares of the message, and then encrypts the i^{th} share under the i^{th} public key. To compute a secret key for identity id, the key generation algorithm first chooses λ different indices $j_1, \ldots, j_\lambda$. It then

computes a signature σ on id, and generates a key for (id, σ) using each of the λ master secret keys with the corresponding indices. The tracing algorithm uses the underlying indexed scheme's trace algorithm to obtain a set of (id, σ) tuples. It then checks if σ is a valid signature on id; if so, it outputs id as a traitor.

Now, suppose an adversary queries for $t(< n)$ secret keys, and outputs a decoding box D. Let $j_{i,k}$ denote the k^{th} index chosen for the i^{th} secret key. With high probability, there exists an index $k^* \in [\lambda]$ such that the set of indices $\{j_{1,k^*}, j_{2,k^*}, \ldots, j_{t,k^*}\}$ are all distinct. As a result, using the correct-tracing guarantee of the underlying tracing scheme for the k^* strand, we can extract at least one tuple (id, σ).

Next, we need to argue the false trace guarantee. This follows mainly from the security of the signature scheme. Suppose an adversary receives a set of keys corresponding to an identity set $\mathcal{I}$, and outputs a decoding box D. If trace outputs an identity $\mathsf{id} \notin \mathcal{I}$, then this means the sub-trace algorithm output a tuple (id, σ) such that σ is a valid signature on id. As a result, σ is a forgery on message id (because the adversary did not query for a key corresponding to id).

2.2.5 Bounded Embedded-Identity TT to Unbounded Embedded-Identity TT

The final component is to transform a tracing system for bounded keys to one with no bound on the number of keys issued. For this transformation

to be efficient, it is essential that the underlying bounded EITT scheme to have ciphertexts with polylogarithmic dependence on the key bound n. The reason is that our core idea is to have λ (bounded) EITT systems running in parallel, where the i^{th} system runs the bounded tracing scheme with bound $n_i = 2^i$, and if the ciphertext size does not scale polylogarithmically with the bound n_i, then this transformation would not work.[13]

More formally, the setup algorithm runs the bounded system's setup λ times, the i^{th} iteration run with bound $n_i = 2^i$. It sets the public key (resp. master secret key and the tracing key) to be the λ public keys (resp. the λ different master secret keys and the tracing keys). The encryption algorithm secret shares the message into λ shares, and encrypts the i^{th} share using the i^{th} public key. The key generation algorithm computes λ different secret keys. Finally, the tracing algorithm runs the bounded system's trace algorithm, one by one, until it finds a traitor. First, note that since the adversary is polynomially bounded, if it queries for t keys, then there exists some i^* such that $t \leq 2^{i^*} < 2t$. As a result, the trace is guaranteed to find a traitor in the i^{*th} system, and hence it runs in time $\mathsf{poly}(2^{i^*}) = \mathsf{poly}(t)$. Second, since every underlying bounded system's false trace guarantee holds even if the adversary queries for more keys than permitted, thus none of the premature sub-traces result in a false trace. At a very high level, the central observation here that allows us in avoiding the need for adaptive security is that: while tracing

[13]Due to similar reasons, it is essential that the running time of all algorithms (except possibly the tracing algorithm) grows at most polylogarithmically with n.

we simply perform the "tighest" fit search for finding the smallest polynomial bound on keys corrupted and then carry out the tracing procedure rather than tracing on an exponential sized space directly. Similar techniques of combining different *bounded adversary* instances, and invoking the security of the instance with *just high enough* security were used previously in [17, 61].

2.3 Some future directions

Our construction of Mixed FE relied on the LWE assumption and leveraged certain algebraic properties in that setting. An intriguing question is whether there are other avenues for achieving Mixed FE. A natural path is to build Mixed FE with a garbled circuits [118] backbone. If one starts with the bounded key functional encryption scheme of Gorbunov, Vaikuntanathan, and Wee (GVW) [75] and flips [4, 35, 92] the semantics of message and function, one can get a secret key functional encryption scheme that is secure for an unbounded number of private keys and bounded number of ciphertexts. To make it a Mixed FE system we would somehow connect a public key mode of encryption to the scheme. One possible path is to use a "blinded" [34] form of garbled circuits as the underlying 1-query scheme in the GVW transformation. (Building on [60, 64], blinded garbled circuits were recently used to give anonymous identity-based encryption (IBE) from new assumptions). It seems possible that this approach could lead to a scheme with the accept indistinguishability property if no encryption oracle queries are allowed. However, there appears to be technical difficulties in making a public key–generated ci-

phertext indistinguishable from a master secret key–generated ciphertext when the attacker gets oracle queries. That being said, we believe that a garbled circuit approach remains a plausible future direction.

We remark that even if a garbled circuit approach becomes possible, the requirement for an ABE scheme supporting circuits will still indirectly require the LWE assumption given the state of the art. In addition, we expect that our LWE toolkit and underlying construction ideas will have future value in any case.

Another interesting direction is whether there are other applications that can leverage a functional encryption system that has a bimodal encryption where the public key and master secret key support different spaces of messages or functions. In our Mixed FE system the public key only supported the always accept function, but there could conceivably be other variants of interest.

Finally, a natural open question is to construct traitor tracing schemes with public traceability from LWE. Currently, it is unclear if achieving public tracing is an easier task than building general public key functional encryption.

2.4 Additional related work

Weaker notions of traitor tracing Since the notion of traitor tracing (TT) was first proposed [47], several relaxed variants have been studied in order to achieve short ciphertexts. The first natural relaxation is the bounded collusion setting, where we have an a priori bound k that is fixed during

setup, and security is guaranteed only if the adversary gets at most k secret keys. Collusion bounded systems can either be constructed via combinatorial tools [47, 116, 48, 115, 106, 14] or be algebraic and constructed under different cryptographic assumptions such as DDH [93, 19, 90, 94], bilinear DDH [41, 1, 65], and LWE [95]. Recently, Agrawal et al. [2] showed a transformation from inner product functional encryption to collusion-bounded TT, resulting in algebraic constructions based on various assumptions such as DCR, DDH, and LWE. In all the above works, the size of the ciphertext grows with the collusion bound.

The second relaxation is called threshold TT, introduced by [101, 48]. In a threshold TT scheme, a threshold $\delta \in [0, 1]$ is chosen during setup, and the traceability guarantee only holds if the decoder box works with probability at least δ. Boneh and Naor [24] showed a threshold TT scheme where the ciphertexts have size $O(\lambda)$ and the secret keys have size $O(n^2\lambda/\delta^2)$. While this scheme achieves collusion resistance, the system must be configured with a specific δ value, and once it is set, one will not necessarily be able to identify a traitor from a box D that works with smaller probability. In practice, it can be tricky to ascertain what threshold will actually be okay. This is because the encrypted messages could have redundancy, so even a decoder box with a small fraction of success might allow an attacker to learn the underlying message.

Finally, in a recent work, Goyal et al. [79] introduced a new relaxation called *risky traitor tracing*. In this notion, the scheme is fully collusion resistant

(and does not have the threshold restriction as above). Instead, the probability of tracing a traitor, given a successful decoding box, can be substantially smaller than 1. For instance, [79] showed a bilinear maps–based construction where the ciphertext size grows as $\lambda \cdot k$, but the trace algorithm has only a k/n chance of catching a traitor. The authors show that this weaker notion is actually enough to achieve strong hardness results for differential privacy [62, 63] and also argue that in a certain "continuous use" setting, the probability of tracing can be amplified back up to one. However, in general settings, the Goyal et al. tradeoff between the probability of catching a traitor and the size of ciphertexts might be undesirable.

Connections to differential privacy Dwork et al. [63] showed that existence of collusion resistant traitor tracing schemes implies hardness results for efficient differentially private [62] data sanitization. In particular, they showed that if there exists a traitor tracing scheme with ciphertexts of size $s(\lambda, n)$, then there exist a database of size n and a query class $\mathcal{Q}$ of size $2^{s(\lambda,n)}$ such that it is hard to sanitize the database D for query class $\mathcal{Q}$ in a differentially private manner. Combining our LWE-based construction with the result of Dwork et al. , we get an LWE-based hardness result for differentially private sanitization with query space of size $2^{\mathsf{poly}(\lambda,\log n)}$. We note that Goyal et al. [79] and Kowalczyk et al. [92] recently achieved better differential privacy impossibility results from the security of bilinear map assumptions and one-way functions, respectively.

Concurrent work. Concurrent to our work, Chen, Vaikuntanathan, and Wee [43] also proposed a simplified variant of our row-removal and target switching properties. They used these properties for constructing lockable obfuscation schemes [80, 117] for nonpermutation branching programs. However, their properties are weaker than the ones we define/construct in this work. In particular, their constructions do not allow the adversary to make any preimage queries.

Combining ABE with other primitives. Our transformation from Mixed FE to PLBE using ABE has some high-level similarities to the predicate encryption scheme of Gorbunov, Vaikuntanathan, and Wee [77] and the single-key functional encryption scheme of Goldwasser et al. [74]. In both these cases, we have specialized encryption schemes whose ciphertext serves as an attribute for the ABE scheme.

Chapter 3

Preliminaries

3.1 Notation

Let PPT denote probabilistic polynomial time. We will use lowercase bold letters for vectors (e.g., $\mathbf{v}$) and uppercase bold letters for matrices (e.g., $\mathbf{A}$), and we assume all vectors are row vectors. The jth row of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}[j]$. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q. We represent $\mathbb{Z}_q$ as integers in the range $(-q/2, q/2]$. For a vector $\mathbf{v}$, we let $\|\mathbf{v}\|$ denote its ℓ_2 norm and $\|\mathbf{v}\|_\infty$ denote its infinity norm. Similarly, for matrices, $\|\cdot\|$ and $\|\cdot\|_\infty$ denote their ℓ_2 and infinity norms, respectively.

We denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. Throughout this paper, unless specified, all polynomials we consider are positive polynomials. Also, we represent each finite set on integers $S \subset \mathbb{N}$ as an *ordered* set $S = \{i_1, i_2, \ldots, i_n\}$, i.e., $i_j < i_k$ for every $1 \leq j < k \leq n$. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each component is drawn independently from the distribution $\mathcal{D}$.

For two distributions X, Y, over a finite domain Ω, the statistical distance between X and Y is defined as $\mathsf{SD}(X,Y) \stackrel{\text{def}}{=} \frac{1}{2}\sum_{\omega\in\Omega}|X(\omega)-Y(\omega)|$. A family of distributions $\mathcal{D}_1 = \{\mathcal{D}_1(\lambda)\}_\lambda$ and $\mathcal{D}_2 = \{\mathcal{D}_2(\lambda)\}_\lambda$, parameterized by security parameter λ, are said to be statistically indistinguishable, represented by $\mathcal{D}_1 \approx_s \mathcal{D}_2$, if there exists a negligible function $\text{negl}(\cdot)$ such that, for all $\lambda \in \mathbb{N}$, $\mathsf{SD}(\mathcal{D}_1(\lambda), \mathcal{D}_2(\lambda)) \leq \text{negl}(\lambda)$. For a family of distributions $\mathcal{D} = \{\mathcal{D}(\lambda)\}_\lambda$ over the integers, and integer bounds $B = \{B(\lambda)\}_\lambda$, we say that $\mathcal{D}$ is B-bounded if $\Pr[|x| \leq B(\lambda) : x \leftarrow \mathcal{D}(\lambda)] = 1$. In words, a B-bounded distribution is supported only on the range $[-B, B]$. Below we state the "smudging" lemma as it appears in prior works.

Lemma 3.1.1 (Smudging Lemma [9, Lemma 2.1, paraphrased])**.** *Let B_1, B_2 be two polynomials over the integers, and let $\mathcal{D} = \{\mathcal{D}(\lambda)\}_\lambda$ be any B_1-bounded distribution family. Let $U = \{U(\lambda)\}_\lambda$ and $U(\lambda)$ denote the uniform distribution over integers $[-B_2(\lambda), B_2(\lambda)]$. The family of distributions $\mathcal{D}$ and U is statistically indistinguishable, $\mathcal{D} + U \approx_s U$, if there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $B_1(\lambda)/B_2(\lambda) \leq negl(\lambda)$.*

3.2 Lattice preliminaries

An m-dimensional lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^m$. Given positive integers n, m, q and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we let $\Lambda_q^\perp(\mathbf{A})$ denote the lattice $\{\mathbf{x} \in \mathbb{Z}^m \ : \ \mathbf{A}\cdot\mathbf{x}^T = \mathbf{0}^T \mod q\}$. For $\mathbf{u} \in \mathbb{Z}_q^n$, we let $\Lambda_q^{\mathbf{u}}(\mathbf{A})$ denote the coset $\{\mathbf{x} \in \mathbb{Z}^m \ : \ \mathbf{A}\cdot\mathbf{x}^T = \mathbf{u}^T \mod q\}$.

Discrete Gaussians Let σ be any positive real number. The Gaussian distribution $\mathcal{D}_\sigma$ with parameter σ is defined by the probability distribution function $\rho_\sigma(\mathbf{x}) = \exp(-\pi \|\mathbf{x}\|^2 / \sigma^2)$. For any set $\mathcal{L} \subset \mathbb{R}^m$, define $\rho_\sigma(\mathcal{L}) = \sum_{\mathbf{x}\in\mathcal{L}} \rho_\sigma(\mathbf{x})$. The discrete Gaussian distribution $\mathcal{D}_{\mathcal{L},\sigma}$ over $\mathcal{L}$ with parameter σ is defined by the probability distribution function $\rho_{\mathcal{L},\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\mathcal{L})$ for all $\mathbf{x} \in \mathcal{L}$.

The following lemma (Lemma 4.4 of [97], [72]) shows that if the parameter σ of a discrete Gaussian distribution is small, then any vector drawn from this distribution will be short (with high probability).

Lemma 3.2.1. *Let m, n, q be positive integers with $m > n$, $q \geq 2$, and $\sigma = \tilde{\Omega}(n)$. Then*

$$\Pr[\|\mathbf{x}\| > \sqrt{m} \cdot \sigma : \mathbf{x} \leftarrow \mathcal{D}_{\mathcal{L},\sigma} : \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathcal{L} = \Lambda_q^\perp(\mathbf{A})] \leq \mathit{negl}(n).$$

Truncated discrete Gaussians The truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ, denoted by $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$, is the same as the discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^m,\sigma}$ except it outputs 0 whenever the ℓ_∞ norm exceeds $\sqrt{m}\cdot\sigma$. Note that, by definition, $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m}\cdot\sigma$-bounded. Also, by the above lemma we get that $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma} \approx_s \mathcal{D}_{\mathbb{Z}^m,\sigma}$.

3.2.1 Learning with errors

The learning with errors (LWE) problem was introduced by Regev [107], who showed that solving LWE on *average* is as hard as quantumly solving several standard lattice-based problems in the worst case. The LWE assumption

states that no polynomial time adversary can distinguish between the following oracles. In one case, the oracle chooses a uniformly random secret $\mathbf{s}$, and for each query, it chooses a vector $\mathbf{a}$ uniformly at random, scalar e from a noise distribution, and outputs $(\mathbf{a}, \mathbf{s} \cdot \mathbf{a}^T + e)$. In the second case, the oracle simply outputs a uniformly random vector $\mathbf{a}$ together with a uniformly random scalar u. Regev showed that if there exists a polynomial time adversary that can break the LWE assumption, then there exists a polynomial time quantum algorithm that can solve some hard lattice problems in the worst case.

Several works also explored different variants of the LWE assumption, where the secret vector $\mathbf{s}$, public vectors $\mathbf{a}$, and noise are drawn from different distributions. In this work, we will be using two of these variants. First, we will be using the LWE version with *short secrets* (also known as the *normal form*), introduced by Applebaum et al. [8]. In this variant, the secret vector $\mathbf{s}$ is also drawn from the noise distribution. Applebaum et al. showed that this version is as hard as the LWE problem if the modulus is p^e for some prime p and integer e. This was later generalized to all moduli by Brakerski et al. [33]. The second variant, which was proposed by Boneh et al. [23], allows the public vectors $\mathbf{a}$ to be chosen from the noise distribution as well. Boneh et al. showed that this version of LWE is as hard as standard LWE.

We will first present the LWE assumption in a general framework,[1]

[1]Canetti and Chen [39] proposed the general LWE problem. However, their version requires the public vectors to be sampled from a uniform distribution, whereas we require the public vectors to be sampled from nonuniform distributions. Also, it is possible to generalize our version further. Here, we present the minimal generalization that suffices for

which captures the standard LWE, LWE with short secrets, and LWE with short public vectors. In this framework, we will have an explicit security parameter λ, and the other parameters are allowed to grow as a function of the security parameter.

Definition 3.2.1 (generalized learning with errors)**.** Fix any polynomial $n(\cdot)$, function $q(\cdot)$, secret distribution $\eta(\cdot)$, public vector distribution $\phi(\cdot)$, and noise distribution $\chi(\cdot)$, where $n : \mathbb{N} \to \mathbb{N}$, $q : \mathbb{N} \to \mathbb{N}$ and for each $\lambda \in \mathbb{N}$, $\eta(\lambda)$ and $\phi(\lambda)$ are distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$ and $\chi(\lambda)$ is a distribution over $\mathbb{Z}$. We say that the generalized LWE assumption $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $n = n(\lambda)$, $\eta = \eta(\lambda)$, $\phi = \phi(\lambda)$, and $\chi = \chi(\lambda)$, $\mathsf{Adv}^{n,q,\eta,\phi,\chi}_{\mathsf{GLWE},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$, where

$$\mathsf{Adv}^{n,q,\eta,\phi,\chi}_{\mathsf{GLWE},\mathcal{A}}(\lambda) = \Pr\left[\mathcal{A}^{O_1^{\mathbf{s}}()}(1^\lambda) = 1 : \mathbf{s} \leftarrow \eta\right] - \Pr\left[\mathcal{A}^{O_2()}(1^\lambda = 1)\right],$$

and oracles $O_1^{\mathbf{s}}(), O_2()$ are defined as follows: oracle $O_1^{\mathbf{s}}()$ has $\mathbf{s} \in \mathbb{Z}_q^n$ hardwired, and on each query it chooses $\mathbf{a} \leftarrow \phi$, $e \leftarrow \chi$ and outputs $\left(\mathbf{a}, \mathbf{s} \cdot \mathbf{a}^T + e \bmod q\right)$, and oracle $O_2()$ (on each query) chooses $\mathbf{a} \leftarrow \phi$, $u \leftarrow \mathbb{Z}_q$ and outputs $(\mathbf{a}, u)$.

We now present different variants of the LWE assumption and discuss the parameters for which they are believed to be secure.

Assumption 1 (learning with errors). Let $n : \mathbb{N} \to \mathbb{N}$ be a polynomial, and let $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ be functions. The $\mathsf{LWE}_{n,q,\sigma}$ assumption states that $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds, where $\eta(\lambda), \phi(\lambda)$ are uniform distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$, and $\chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma(\lambda)}$.

---

our work.

The following theorem shows that breaking LWE is as hard as solving hard lattice problems. In particular, given the current state of the art of lattice problems, the LWE assumption is believed to be true for any polynomial $n(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, the following constraints are satisfied: $0 < \sigma < q < 2^n$, $n \cdot q/\sigma < 2^{n^\epsilon}$ (for any constant $\epsilon < 1$), and $\sigma > 2\sqrt{n}$.

Theorem 3.2.1 (LWE to worst-case lattice problem [107, 105, 33])**.** *Fix any polynomial $n(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, $0 < \sigma < q < 2^n$, and $\sigma > 2\sqrt{n}$. For every $\lambda \in \mathbb{N}$, let $\eta = \eta(\lambda)$ and $\phi = \phi(\lambda)$ denote the uniform distributions over $\mathbb{Z}_q^n$ and $\chi = \chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma}$. If there exist a PPT algorithm $\mathcal{A}$ and a nonnegligible function $\epsilon_{\mathcal{A}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{n,q,\eta,\phi,\chi}_{\mathsf{GLWE},\mathcal{A}}(\lambda) \geq \epsilon_{\mathcal{A}}(\lambda)$, then there exist a PPT algorithm $\mathcal{B}$ and a nonnegligible function $\epsilon_{\mathcal{B}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all instances X of $\mathsf{GapSVP}_{n,n\cdot q/\sigma}$, $\mathcal{B}$ can solve X with probability at least $\epsilon_{\mathcal{B}}(\lambda)$.*

Assumption 2 (LWE with short secrets). Let $n : \mathbb{N} \to \mathbb{N}$ be a polynomial, and let $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ be functions. The $\mathsf{LWE\text{-}ss}_{n,q,\sigma}$ assumption states that $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds, where $\phi(\lambda)$ is the uniform distributions over $\mathbb{Z}^{n(\lambda)}_{q(\lambda)}$, $\eta(\lambda) \equiv \mathcal{D}_{\mathbb{Z}^{n(\lambda)},\sqrt{2}\sigma(\lambda)}$, and $\chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma(\lambda)}$.

The next theorem shows that breaking LWE with short secrets is as hard as breaking (standard) LWE, provided $0 < \sigma(\lambda) < q(\lambda) < 2^{n(\lambda)}$ and $\sigma(\lambda) > \lambda$.

Theorem 3.2.2 (LWE with short secrets [8, Lemma 2], [33, Lemma 2.12])**.** *Fix any polynomial $n(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, $0 < \sigma < q < 2^n$, and $\sigma > \lambda$.*[2] *For every $\lambda \in \mathbb{N}$, let $\eta(\lambda) \equiv \mathcal{D}_{\mathbb{Z}_q^n, \sqrt{2}\sigma}$, let $\phi(\lambda)$ be the uniform distribution over $\mathbb{Z}_q^n$, and let $\chi(\lambda) \equiv \mathcal{D}_{\mathbb{Z},\sigma}$. If there exist a PPT algorithm $\mathcal{A}$ and a nonnegligible function $\epsilon_{\mathcal{A}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{n,q,\eta,\phi,\chi}_{\mathsf{GLWE},\mathcal{A}}(\lambda) \geq \epsilon_{\mathcal{A}}(\lambda)$, then there exist a PPT algorithm $\mathcal{B}$ and a nonnegligible function $\epsilon_{\mathcal{B}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{n,q,\phi,\phi,\chi}_{\mathsf{GLWE},\mathcal{B}}(\lambda) \geq \epsilon_{\mathcal{B}}(\lambda)$.*

Assumption 3 (LWE with short public vectors). Let $n : \mathbb{N} \to \mathbb{N}$ be a polynomial, let $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ be functions, and let $\{\chi(\lambda)\}_{\lambda \in \mathbb{N}}$ be the family of distributions over $\mathbb{Z}$. The $\mathsf{LWE\text{-}sp}_{n,q,\sigma,\chi}$ assumption states that $\mathsf{GLWE}_{n,q,\eta,\phi,\chi}$ holds, where $\eta(\lambda)$ is the uniform distributions over $\mathbb{Z}_{q(\lambda)}^{n(\lambda)}$, $\phi(\lambda) \equiv \mathcal{D}_{\mathbb{Z}^{n(\lambda)},\sigma(\lambda)}$.

The last theorem in this sequence shows a reduction from LWE with short public vectors to (standard) LWE with a lower dimension.

Theorem 3.2.3 (LWE with short public vectors [23, Corollary 4.6])**.** *Fix any polynomials $n(\cdot), k(\cdot)$ and functions $q(\cdot), \sigma(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $q = q(\lambda)$, $k = k(\lambda)$, $\sigma = \sigma(\lambda)$, $0 < \sigma < q < 2^n$, $k \geq 6n \log q$, and $\sigma \geq \sqrt{n \log q}$. For every $\lambda \in \mathbb{N}$, let $\phi(\lambda) \equiv \mathcal{D}_{\mathbb{Z}_q^k,\sigma}$, and let $\eta(\lambda), \phi'(\lambda)$ denote the uniform distributions over $\mathbb{Z}_q^k$ and $\mathbb{Z}_q^n$, respectively. Now for any distribution $\chi(\lambda)$ over $\mathbb{Z}$, if there exist a PPT algorithm $\mathcal{A}$ and a nonnegligible function $\epsilon_{\mathcal{A}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{k,q,\eta,\phi,\chi}_{\mathsf{GLWE},\mathcal{A}}(\lambda) \geq \epsilon_{\mathcal{A}}(\lambda)$, then there exist*

[2] Strictly speaking, it is only required that $\sigma > \sqrt{\ln n + \omega(1) \ln \lambda}$.

a PPT algorithm $\mathcal{B}$ and a nonnegligible function $\epsilon_{\mathcal{B}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{n,q,\phi',\phi',\chi}_{\mathsf{GLWE},\mathcal{B}}(\lambda) \geq \epsilon_{\mathcal{B}}(\lambda)$.

3.2.2 Lattice trapdoors

Lattices with trapdoors are lattices that are indistinguishable from randomly chosen lattices, but have certain "trapdoors" that allow efficient solutions to hard lattice problems.

A trapdoor lattice sampler [6, 72] consists of algorithms $\mathsf{TrapGen}$ and $\mathsf{SamplePre}$ with the following syntax and properties:

- $\mathsf{TrapGen}(1^n, 1^m, q) \rightarrow (\mathbf{A}, T_{\mathbf{A}})$: The lattice generation algorithm is a randomized algorithm that takes as input the matrix dimensions n, m, modulus q, and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $T_{\mathbf{A}}$.

- $\mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{u}) \rightarrow \mathbf{s}$: The presampling algorithm takes as input a matrix $\mathbf{A}$, trapdoor $T_{\mathbf{A}}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a parameter $\sigma \in \mathbb{R}$ (which determines the length of the output vectors).[3] It outputs a vector $\mathbf{s} \in \mathbb{Z}_q^m$ such that $\mathbf{A} \cdot \mathbf{s}^T = \mathbf{u}^T$ and $\|\mathbf{s}\| \leq \sqrt{m} \cdot \sigma$.

We require these algorithms to satisfy the following well-sampledness properties. While these properties are similar in spirit to the ones in previous

[3]Note that the preimage sampling algorithm could be easily generalized to generate preimages of matrices in $\mathbb{Z}_q^{n \times k}$ (for any k) by independently running $\mathsf{SamplePre}$ algorithm on each column of the matrix. Throughout this work, we overload the notation by directly giving matrices $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$ as inputs to the $\mathsf{SamplePre}$ algorithm.

works on lattice trapdoors [6, 72, 96], there are a couple of differences. First, we present these properties as a security game between a challenger and a computationally bounded adversary.[4] Second, we separate out the dimensions of the matrix and the security parameter.

The first property (well-sampledness of matrix) states that the matrix output by $\mathsf{TrapGen}$ should look like a uniformly random matrix.

Definition 3.2.2 (well-sampledness of matrix)**.** Fix any function $q : \mathbb{N} \to \mathbb{N}$. A pair of trapdoor generation algorithms $\mathcal{T} = (\mathsf{TrapGen}, \mathsf{SamplePre})$ is said to satisfy the *q-well-sampledness of matrix* property if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\mathsf{pr}^{\mathrm{matrix},q}_{\mathcal{T},\mathcal{A}}(\lambda) = \Pr[1 \leftarrow \mathsf{Expt}^{\mathrm{matrix},q}_{\mathcal{T},\mathcal{A}}(\lambda)] \leq 1/2 + \mathrm{negl}(\lambda)$, where $\mathsf{Expt}^{\mathrm{matrix},q}_{\mathcal{T},\mathcal{A}}(\lambda)$ is defined in Figure 3.1.

$\mathsf{Expt}^{\mathrm{matrix},q}_{\mathcal{T},\mathcal{A}}(\lambda)$

1. Adversary $\mathcal{A}$ receives input 1^λ and sends $1^n, 1^m$ such that $m > n \log q(\lambda) + \lambda$.
2. Challenger chooses $b \leftarrow \{0,1\}$ and $(\mathbf{A}_0, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ and $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times m}$. It sends $\mathbf{A}_b$ to the adversary.
3. $\mathcal{A}$ outputs its guess b'. The experiment outputs 1 iff $b = b'$.

Figure 3.1: Experiment $\mathsf{Expt}^{\mathrm{matrix},q}_{\mathcal{T},\mathcal{A}}$

The next property states that the preimage of a uniformly random vec-

[4] In some cases, we can consider computationally unbounded adversaries if the inputs of the adversary are polynomially bounded.

tor/matrix is indistinguishable from a matrix with entries drawn from Gaussian distribution.

Definition 3.2.3 (preimage sampling)**.** Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{N}$. A pair of trapdoor generation algorithms $\mathcal{T} = (\mathsf{TrapGen}, \mathsf{SamplePre})$ is said to satisfy the (q, σ)-*preimage sampling* property if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, $\mathsf{pr}^{\mathrm{preimg},q,\sigma}_{\mathcal{T},\mathcal{A}}(\lambda) = \Pr[1 \leftarrow \mathsf{Expt}^{\mathrm{preimg},q,\sigma}_{\mathcal{T},\mathcal{A}}(\lambda)] \leq 1/2 + \mathrm{negl}(\lambda)$, where $\mathsf{Expt}^{\mathrm{preimg},q,\sigma}_{\mathcal{T},\mathcal{A}}(\lambda)$ is defined in Figure 3.2.

$\mathsf{Expt}^{\mathrm{preimg},q,\sigma}_{\mathcal{T},\mathcal{A}}(\lambda)$

1. Adversary $\mathcal{A}$ receives input 1^λ and sends $1^n, 1^m, 1^k$ such that $\sigma(\lambda) > \sqrt{n \cdot \log q \cdot \log m} + \lambda$ and $m > n \log q(\lambda) + \lambda$.
2. Challenger chooses $b \leftarrow \{0,1\}$ and $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$; $\mathbf{Z} \leftarrow \mathbb{Z}_q^{n \times k}$, $\mathbf{U}_0 \leftarrow \mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{Z})$ and $\mathbf{U}_1 \leftarrow \mathcal{D}^{m \times k}_{\mathbb{Z},\sigma}$. It sends $(\mathbf{A}, \mathbf{U}_b)$ to the adversary.
3. $\mathcal{A}$ outputs its guess b'. The experiment outputs 1 iff $b = b'$.

Figure 3.2: Experiment $\mathsf{Expt}^{\mathrm{preimg},q,\sigma}_{\mathcal{T},\mathcal{A}}$

These properties are satisfied by the trapdoor-based preimage samplers of [72, 96].

3.3 Branching programs

Branching programs are a model of computation used to capture space-bounded computations [30, 13]. In this work, we will be working with *leveled* branching programs.

Definition 3.3.1 (leveled branching program)**.** A leveled branching program of length L, width w, and input space $\{0,1\}^n$ consists of a sequence of $2L$ functions $\pi_{i,b} : [w] \to [w]$ for $1 \leq i \leq L, b \in \{0,1\}$, an input selection function $\mathsf{inp} : [L] \to [n]$, an accepting state $\mathsf{acc} \in [w]$, and a rejection state $\mathsf{rej} \in [w]$. The starting state st_0 is set to be 1 without loss of generality. The branching program evaluation on input $x \in \{0,1\}^n$ proceeds as follows:

- For $i = 1$ to L,
  - Let $\mathsf{pos} = \mathsf{inp}(i)$ and $b = x_{\mathsf{pos}}$. Compute $\mathsf{st}_i = \pi_{i,b}(\mathsf{st}_{i-1})$.
- If $\mathsf{st}_L = \mathsf{acc}$, output 1. If $\mathsf{st}_L = \mathsf{rej}$, output 0, else output $\perp$.

Additionally, we also define a notion of "input-circling" (leveled) branching programs. In an input-circling branching program, the input bits are read sequentially in ascending order (i.e., $1, \ldots, n, 1, \ldots$). Thus, the input-selector function inp is fixed. Additionally, each bit must be read the same number of times. Formally, we describe this as follows.

Definition 3.3.2. A branching program $\mathsf{BP} = \big(\{\pi_{i,b} : [w] \to [w]\}_{i \in [L], b \in \{0,1\}}, \mathsf{acc} \in [w], \mathsf{rej} \in [w] \big)$ with input space $\{0,1\}^n$ is said to be a input-circling branching program if for all $i \leq L$, $\mathsf{inp}(i) = ((i-1) \bmod n)+1$, and $L \bmod n = 0$.

Any leveled branching program of length L and input space $\{0,1\}^n$ can be easily transformed to an input-circling branching program of length $n \cdot L$. Here, we work with classes of branching programs that all share the same

input selector function $\mathsf{inp}(\cdot)$ which is known during setup. The input selector as described above is just one possibility, and we stick with it for simplicity. Note that we do not require the transition functions $\pi_{i,b}$ to be permutations.

3.4 Public-key encryption and signatures

Public-key encryption A public key encryption scheme PKE with message space $\mathcal{M}$ consists of three algorithms Setup, Enc and Dec with the following syntax:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ The setup algorithm takes as input the security parameter 1^λ and outputs a public key pk and secret key sk.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}) \to \mathsf{ct}$ The encryption algorithm takes as input a public key pk and a message $m \in \mathcal{M}$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to x \in \mathcal{M} \cup \{\perp\}$ The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $x \in \mathcal{M} \cup \{\perp\}$.

Correctness For correctness, we require that for all security parameters λ, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and messages $m \in \mathcal{M}$, $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Definition 3.4.1 ($\mathsf{IND\text{-}CPA}$ Security)**.** A public key encryption scheme $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ is said to be $\mathsf{IND\text{-}CPA}$ secure if for all security parameters λ, stateful PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$ is negligible in λ, where advantage

of $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = \Pr\left[\mathcal{A}(\mathsf{ct}) = b : \begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda);\ b \leftarrow \{0,1\} \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk});\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array}\right].$$

Signature schemes A signature scheme $\mathcal{S} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ with message space $\mathcal{M}$ consists of three algorithms, as follows:

$\mathsf{Setup}(1^\lambda)$ is a randomized algorithm that takes security parameter λ as input and returns a pair of keys $(\mathsf{sk}, \mathsf{vk})$, where sk is the signing key and vk is the verification key.

$\mathsf{Sign}(\mathsf{sk}, m)$ is a possibly randomized algorithm that takes as input the signing key sk, and a message m, and returns a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ is a determistic algorithm that takes as input the verification key vk, a message m, and a signature σ, and outputs 1 (accepts) if verification succeeds, and 0 (rejects) otherwise.

Correctness A signature scheme $\mathcal{S}$ must satisfy the following correctness requirement: For all $\lambda \in \mathbb{N}$, $m \in \mathcal{M}$, and signing/verification keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$

$$\mathsf{Verify}\left(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)\right) = 1.$$

Definition 3.4.2. A signature scheme $\mathcal{S} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ is a secure signature scheme if for every PPT attacker $\mathcal{A}$ there exists a negligible function

negl($\cdot$) such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}^{\mathcal{S}}_{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$, where advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}^{\mathcal{S}}_{\mathcal{A}}(\lambda) = \Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 : \begin{array}{c}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m^*, \sigma^*) = \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{vk})\end{array}\right],$$

and $\mathcal{A}$ should never have queried m^* to Sign oracle.

3.5 Key-policy attribute-based encryption

A key-policy attribute-based encryption (KP-ABE) scheme ABE, for a set of attribute spaces $\mathcal{X} = \{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\mathcal{C} = \{\mathcal{C}_\kappa\}_\kappa$, and message spaces $\mathcal{M} = \{\mathcal{M}_\kappa\}_\kappa$, consists of four polytime algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and a functionality index κ. It outputs the public parameters pp and the master secret key msk.

- $\mathsf{Enc}(\mathsf{pp}, x, m) \to \mathsf{ct}$. The encryption algorithm takes as input public parameters pp, an attribute $x \in \mathcal{X}_\kappa$, and a message $m \in \mathcal{M}_\kappa$. It outputs a ciphertext ct.

- $\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$. The key generation algorithm takes as input master secret key msk and a predicate $C \in \mathcal{C}_\kappa$. It outputs a secret key sk_C.

- $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \to m$ or $\perp$. The decryption algorithm takes as input a secret

key sk_C and a ciphertext ct. It outputs either a message $m \in \mathcal{M}_\kappa$ or a special symbol $\perp$.

Correctness A KP-ABE scheme is said to be correct if there exist negligible functions $\mathrm{negl}_1(\cdot), \mathrm{negl}_2(\cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, for all $x \in \mathcal{X}_\kappa$, $C \in \mathcal{C}_\kappa$, $m \in \mathcal{M}_\kappa$, the following holds:

$$C(x) = 1 \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C); \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m)\end{array}\right] \geq 1 - \mathrm{negl}_1(\lambda),$$

$$C(x) = 0 \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = \perp \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C); \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m)\end{array}\right] \geq 1 - \mathrm{negl}_2(\lambda).$$

Security The standard notion of security for a KP-ABE scheme is that of full or adaptive security. It is formally defined as follows.

Definition 3.5.1. A KP-ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be fully secure if for every stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c}1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ ((m_0, m_1), x) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp}); \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m_b)\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where every predicate query C, made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle, must satisfy the condition that $C(x) = 0$.

In this work, we only require the scheme to achieve selective security, which is formally defined as follows.

Definition 3.5.2. A KP-ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be selectively secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c}(1^\kappa, x) \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp}); \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, x, m_b)\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where every predicate query C, made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle, must satisfy the condition that $C(x) = 0$.

Chapter 4

Traitor tracing and Mixed functional encryption

In this chapter, we first present the syntax and security definitions for traitor tracing schemes. We follow that by introducing the notion of mixed functional encryption (Mixed FE) which is the most crucial component towards building traitor tracing systems.

4.1 Traitor tracing

The notion of traitor tracing was introduced by Chor, Fiat, and Naor [47]. In a TT scheme for n parties, the setup algorithm chooses a master secret key, a public key, and n secret keys for the users. Encryption can be performed using the public key, and each user can decrypt the ciphertext using his/her secret key. There is also a trace algorithm that, given black box access to a successful pirate decoding box, can catch the traitors who colluded to create the pirate decoding box. Traditional definitions of traitor tracing [47, 25] required that the trace algorithm must catch a traitor if a pirate decoding box can decrypt an encryption of a random ciphertext. In this work, we will be using the *indistinguishability*-based definition introduced by Goyal et al. [79],

which is itself based on the definition introduced by Nishimaki, Wichs, and Zhandry [104]. In this definition, the trace algorithm must catch a traitor even if the pirate decoder box can only distinguish between encryptions of two adversarially chosen messages.

Public key traitor tracing A traitor tracing scheme $\mathcal{T}$ with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ consists of four PPT algorithms, $\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}$, and Trace, with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$. The setup algorithm takes as input the security parameter λ and number of users n and outputs a master secret key msk, a public key pk, and n secret keys $\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n$.
- $\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk and message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to y$. The decryption algorithm takes as input a secret key sk and ciphertext ct and outputs $y \in \mathcal{M}_\lambda \cup \{\perp\}$.
- $\mathsf{Trace}^D(\mathsf{msk}, 1^y, m_0, m_1) \to T$. The trace algorithm has oracle access to a program D; it takes as input a master secret key msk, parameter y (in unary), and two messages m_0, m_1. It outputs a set $T \subset \{1, 2, \ldots, n\}$.

Correctness Informally, a correctness requirement states that decrypting an encryption of message m using any one of the valid secret keys must output m. Formally, a TT scheme is said to be correct if there exists a negligible

function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in \{1, 2, \ldots, n\}$, the following holds:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk}, \mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Security There are two security requirements for a TT scheme. First, it is required that it satisfy IND-CPA security. Second, it is required that the tracing algorithm must (almost always) correctly trace at least one key used to create a pirate decoding box (whenever the pirate box successfully decrypts with noticeable probability) and also should not falsely accuse any user of cheating. The formal definitions are provided below.

Definition 4.1.1 (IND-CPA security)**.** A TT scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ is IND-CPA secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \ : \ \begin{array}{c}1^n \leftarrow \mathcal{A}(1^\lambda); b \leftarrow \{0,1\}; \\ (\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Definition 4.1.2 (ind-secure traitor tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a TT scheme. For any nonnegligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Figure 4.1.

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are functions of λ, parameterized by $\mathcal{A}, \epsilon$):

Experiment $\mathsf{Expt\text{-}TT}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^n \leftarrow \mathcal{A}(1^\lambda)$.
- $(\mathsf{msk}, \mathsf{pk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$.
- $T \leftarrow \mathsf{Trace}^D(\mathsf{msk}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$.

Here, $O(\cdot)$ is an oracle that has $\{\mathsf{sk}_i\}_{i\in[n]}$ hardwired, takes as input an index $i \in [n]$, and outputs sk_i. Let S be the set of indices queried by $\mathcal{A}$.

Figure 4.1: Experiment Expt-TT

- Good-Decoder: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$.

  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder}]$.

- Cor-Tr: $T \neq \emptyset \wedge T \subseteq S$.

  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

- Fal-Tr: $T \not\subseteq S$.

  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A TT scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$, and nonnegligible function $\epsilon(\cdot)$, there exist negligible functions $\mathrm{negl}_1(\cdot)$, $\mathrm{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$ the following hold:

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathrm{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}_2(\lambda).$$

Remark 4.1.1 (ind-secure tracing implies IND-CPA). We would like to point out that the if a traitor tracing scheme satisfies ind-secure tracing property

as described in Definition 4.1.2, then it also satisfies the IND-CPA security requirement (Definition 4.1.1). This follows from a simple reduction. Suppose that there exists a successful IND-CPA attacker $\mathcal{A}$ on the TT scheme $\mathcal{T}$, then we could use $\mathcal{A}$ to contradict the correct tracing requirement of the TT scheme. The idea is simply to use the IND-CPA attacker $\mathcal{A}$ to play the secure tracing game, and after the challenge phase when $\mathcal{A}$ outputs a pair of challenge messages m_0 and m_1, then the reduction algorithm outputs the attacker $\mathcal{A}$ itself as its decoding box (i.e., the stateful attacker $\mathcal{A}$ which takes as input a challenge ciphertext and outputs a guess bit is set as the decoding box). Now since $\mathcal{A}$ wins the IND-CPA game with nonnegligible advantage, thus it is a good decoder. Note that since the reduction algorithm does not query for any secret keys (i.e., $S = \emptyset$), thus the probability that the event $\mathsf{Cor\text{-}Tr}$ occurs is exactly 0. Therefore, by security of tracing, the probability that $\mathcal{A}$ is a good decoder is at most negligible. Hence, IND-CPA security of $\mathcal{T}$ follows from the ind-secure traitor tracing property.

4.2 Mixed functional encryption

A regular functional encryption scheme [112, 26] consists of a setup, an encryption, a key generation, and a decryption algorithm. The setup algorithm takes the security parameter and functionality index as inputs, and outputs public parameters and a master secret key. The encryption algorithm uses the public parameters to encrypt a message, while the key generation algorithm uses the master secret key to compute a secret key corresponding to a function. The decryption algorithm takes as input a ciphertext and a secret key, and outputs the function evaluation on the message.

Here we introduce the notion of mixed Functional Encryption (Mixed FE). A Mixed FE scheme is defined as a dual of the standard functional encryption (i.e., ciphertext-policy) in which the secrets keys are associated with a message, and ciphertexts are associated with (boolean) functions. Additionally, in a Mixed FE system, there are two encryption algorithms: Enc and $\mathsf{SK\text{-}Enc}$. The normal encryption algorithm Enc takes as input only the public parameters and outputs an encryption of a "canonical" *always-accepting* function. The "secret key" encryption algorithm, on the other hand, takes as input the master secret key and a function f and encrypts f. The decryption algorithm in a Mixed FE system works similarly to that in standard functional encryption; that is, it outputs the evaluation of encrypted function f on the message m associated with the secret key. Below we provide a formal definition.

Consider function classes $\mathcal{F} = \{\mathcal{F}_\kappa\}_\kappa$ and message spaces $\mathcal{M} = \{\mathcal{M}_\kappa\}_\kappa$,

where $f : \mathcal{M}_\kappa \to \{0,1\}$ for each $f \in \mathcal{F}_\kappa$.[1] A mixed FE scheme Mixed-FE, for function classes $\mathcal{F}$ and message spaces $\mathcal{M}$, consists of five polytime algorithms (Setup, Enc, SK-Enc, KeyGen, Dec) with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and functionality index κ and outputs the public parameters pp and the master secret key msk.
- $\mathsf{Enc}(\mathsf{pp}) \to \mathsf{ct}$. The normal encryption algorithm takes as input public parameters pp and outputs a ciphertext ct.
- $\mathsf{SK\text{-}Enc}(\mathsf{msk}, f) \to \mathsf{ct}$. The secret key encryption algorithm takes as input master secret key msk and a function $f \in \mathcal{F}_\kappa$. It outputs a ciphertext ct.
- $\mathsf{KeyGen}(\mathsf{msk}, m) \to \mathsf{sk}_m$. The key generation algorithm takes as input master secret key msk and a message/input $m \in \mathcal{M}_\kappa$. It outputs a secret key sk_m.
- $\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) \to \{0,1\}$. The decryption algorithm takes as input a secret key sk_m and a ciphertext ct, and it outputs a single bit.

Correctness A mixed FE scheme is said to be correct if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$ and for every $f \in \mathcal{F}_\kappa$, $m \in \mathcal{M}_\kappa$,

[1] The following definition could be easily generalized for multibit function classes, but for simplicity we stick to boolean functions.

the following hold:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) = 1 : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa);\\ \mathsf{sk}_m \leftarrow \mathsf{KeyGen}(\mathsf{msk}, m);\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp})\end{array}\right] \geq 1 - \mathrm{negl}(\lambda),$$

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) = f(m) : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa);\\ \mathsf{sk}_m \leftarrow \mathsf{KeyGen}(\mathsf{msk}, m);\\ \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Security Informally, for security we require that no PPT adversary should be able to distinguish between secret key encryptions of two functions f_0 and f_1 if for every key in its possession, the output of f_0, f_1 is identical. Additionally, we also require that it should be hard to distinguish between normal encryptions and secret key encryptions of the special always-accepting function. In this work, we are only interested in mixed FE schemes that guarantee security against adversaries which make a bounded number of secret key encryption queries. Below we formally define it.

Definition 4.2.1 (q-query function indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed FE scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-query function indistinguishability security if for every stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c}1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa);\\ (f^{(0)}, f^{(1)}) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp});\\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^{(b)})\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle; and
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^{(0)}(m) = f^{(1)}(m)$.

We also define a restricted version of the function indistinguishability game in which the adversary must declare its challenge functions $(f^{(0)}, f^{(1)})$ at the beginning, and it must make all its q encryption queries before any of its key generation queries.

Definition 4.2.2 (q-query restricted function indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed FE scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-query selective function indistinguishability security if for every stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp},\mathsf{ct}) = b : \begin{array}{c}(1^\kappa, f^{(0)}, f^{(1)}) \leftarrow \mathcal{A}(1^\lambda);\\ (\mathsf{pp},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa);\\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^{(b)})\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to the $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle;
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^{(0)}(m) = f^{(1)}(m)$; and

- $\mathcal{A}$ must make all (at most $q(\lambda)$) $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle queries before making any query to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle.

Definition 4.2.3 (q-query accept indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed FE scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-query accept indistinguishability security if for every stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ f^* \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp}); \\ b \leftarrow \{0,1\};\ \mathsf{ct}_1 \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^*); \\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}) \end{array}\right]$$
$$\leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where

- $\mathcal{A}$ can make at most q queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \cdot)$ oracle; and
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy the condition that $f^*(m) = 1$.

Additionally, we also define a restricted notion of the accept indistinguishability property for mixed FE schemes, in which the adversary must declare its challenge function f^* at the beginning, and it must make all its q encryption queries before any of its key generation queries, and it is restricted to only making ciphertext queries for functions f such that all queried f's evaluate to 1 on all (secret key) queried messages m. Below we formally describe it.

Definition 4.2.4 (q-query restricted accept indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed FE scheme $\mathsf{Mixed\text{-}FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{SK\text{-}Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to satisfy q-query *restricted* accept indistinguishability security if for every stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)}(\mathsf{pp},\mathsf{ct}_b) = b : \begin{array}{c}(1^\kappa, f^*) \leftarrow \mathcal{A}(1^\lambda);\\ (\mathsf{pp},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa);\\ b \leftarrow \{0,1\};\ \mathsf{ct}_1 \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^*);\\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp})\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to $\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)$ oracle;
- every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk},\cdot)$ oracle must satisfy the condition that $f^*(m) = 1$ as well as $f(m) = 1$ for every ciphertext query f made by $\mathcal{A}$ to the $\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)$ oracle; and
- $\mathcal{A}$ must make all (at most $q(\lambda)$) $\mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)$ oracle queries before making any query to the $\mathsf{KeyGen}(\mathsf{msk},\cdot)$ oracle.

Chapter 5

Traitor tracing from Mixed FE and ABE via PLBE

In this chapter, we describe our approach towards building a traitor tracing scheme. To that end, we first recall the notion of private linear broadcast encryption (PLBE) [25] as a useful intermediate primitive, and show how to construct TT schemes from PLBE schemes that achieve 1-query security. Our construction is divided into two components, where we first show how to construct a TT scheme from PLBE schemes that achieve decoder-based security. We follow that by showing that PLBE schemes that achieve 1-query security also satisfy decoder-based security. Finally, we construct a 1-query secure PLBE scheme from a key-policy attribute-based encryption (KP-ABE) scheme and a 1 -query secure Mixed FE scheme, thereby constructing a TT scheme using the same primitives.

5.1 Private linear broadcast encryption

Private linear broadcast encryption (PLBE) was introduced by Boneh, Sahai, and Waters [25] as a framework for constructing TT schemes. There are four algorithms in a PLBE scheme: $\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec}$. The setup

algorithm outputs a master secret key, public parameters, and n secret keys, one for each user in the system. The public key encryption algorithm can be used to encrypt messages, and ciphertexts can be decrypted using one of the n secret keys via the decryption algorithm. In addition to these algorithms, there is also a special *trace-encryption* algorithm. This algorithm, which uses the master secret key, can be used to encrypt messages to any index $i \in \{0, 1, \ldots, n\}$. A secret key for user j can decrypt a ciphertext for index i only if $j > i$.

Boneh, Sahai, and Waters [25] proposed three security definitions for PLBE schemes. The first one requires that special encryptions to index 0 must be indistinguishable from public key encryptions, even if the adversary has all the secret keys. The next security requirement is that special encryptions to index $i - 1$ must be indistinguishable from special encryptions to index i if the adversary does not have a secret key for user i. Finally, the third security property is that special encryption of message m_0 to index n must be indistinguishable from special encryption of message m_1 to index n, even if the adversary has all secret keys. However, as discussed in Section 2.1, the BSW definitions of PLBE do not suffice for constructing TT schemes. Here, we first provide the PLBE syntax, and then present the decoder-based and query-based security definitions of PLBE.

Syntax A PLBE scheme $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ has the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{msk}, \mathsf{pp}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$. The setup algorithm takes as input the security parameter λ and number of users n and outputs public parameters pp, master secret key msk, and n secret keys $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$.

- $\mathsf{Enc}(\mathsf{pp}, m) \to \mathsf{ct}$. The encryption algorithm takes as input public parameters pp and message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.

- $\mathsf{Enc\text{-}index}(\mathsf{msk}, m, i \in \{0, 1, 2, \ldots, n\}) \to \mathsf{ct}$. The index-encryption algorithm takes as input the master secret key msk, message $m \in \mathcal{M}_\lambda$, and index $i \in \{0, 1, 2, \ldots, n\}$ and outputs a ciphertext ct.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to y$. The decryption algorithm takes as input a secret key sk and ciphertext ct and outputs $y \in \mathcal{M}_\lambda \cup \{\perp\}$.

Correctness A PLBE scheme is said to be correct if there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in \{0, 1, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, the following hold:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk}, \mathsf{pp}, \{\mathsf{sk}_k\}_{k\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m)\end{array}\right] \geq 1 - \mu(\lambda),$$

$$i < j \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk}, \mathsf{pp}, \{\mathsf{sk}_k\}_{k\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i)\end{array}\right] \geq 1 - \mu(\lambda),$$

$$i \geq j \Rightarrow \Pr\left[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}) = \perp : \begin{array}{c}(\mathsf{msk}, \mathsf{pp}, \{\mathsf{sk}_k\}_{k\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i)\end{array}\right] \geq 1 - \mu(\lambda).$$

5.1.1 *q*-query PLBE security

In this section we extend the existing PLBE security definitions by allowing the adversary to make a bounded number of index-encryption queries.

Below we describe them in detail.

Definition 5.1.1 (q-query normal hiding security)**.** Let $q(\cdot)$ be any fixed polynomial. A PLBE scheme is said to satisfy q-query normal hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $p_{\mathcal{A}}^{q,\mathsf{nrml}}(\lambda) \leq \frac{1}{2} + \mathrm{negl}(\lambda)$, where $p_{\mathcal{A}}^{q,\mathsf{nrml}}(\lambda)$ is defined as follows:

$$\Pr\left[\mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,0)}(\mathsf{ct}_b) = b \;:\; \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ m \leftarrow \mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,0)}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}\right); \\ b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right],$$

where $\mathcal{A}$ can make at most $q(\lambda)$ queries to the $\mathsf{Enc\text{-}index}(\mathsf{msk}, \cdot, 0)$ oracle. Note that here $\mathcal{A}$ is only allowed to query for ciphertexts corresponding to index 0.

Definition 5.1.2 (q-query index hiding security)**.** Let $q(\cdot)$ be any fixed polynomial. A PLBE scheme is said to satisfy q-query index hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$ and every index $i^* \in \{0, \ldots, n-1\}$, $p_{\mathcal{A}}^{q,\mathsf{ind}}(\lambda, i^*) \leq \frac{1}{2} + \mathrm{negl}(\lambda)$, where $p_{\mathcal{A}}^{q,\mathsf{ind}}(\lambda, i^*)$ is defined as follows:

$$\Pr\left[\mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}) = b \;:\; \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ m \leftarrow \mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\{1,\ldots,n\}\setminus\{i^*+1\}}\right); \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i^* + b) \end{array}\right],$$

where $\mathcal{A}$ can make at most $q(\lambda)$ queries to the $\mathsf{Enc\text{-}index}(\mathsf{msk}, \cdot, \cdot)$ oracle. Note that here $\mathcal{A}$ can query the encryption oracle on arbitrary message-index pairs.

Definition 5.1.3 (q-query message hiding security)**.** Let $q(\cdot)$ be any fixed polynomial. A PLBE scheme is said to satisfy q-query message hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, $p_{\mathcal{A}}^{q,\mathsf{msg}}(\lambda) \leq \frac{1}{2} + \mathrm{negl}(\lambda)$, where $p_{\mathcal{A}}^{q,\mathsf{msg}}(\lambda)$ is defined as follows:

$$\Pr\left[\mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}(\mathsf{ct}) = b \ : \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}\right); \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m_b, n) \end{array}\right],$$

where $\mathcal{A}$ can make at most $q(\lambda)$ queries to the $\mathsf{Enc\text{-}index}(\mathsf{msk},\cdot,\cdot)$ oracle. Note that here $\mathcal{A}$ can query the encryption oracle on arbitrary message-index pairs.

5.1.2 Decoder-based PLBE security

In this section we introduce new decoder-based security definitions for PLBE schemes. We start by formally defining the notion of good distinguishers for PLBE schemes w.r.t. different encryption modes.

PLBE distinguishers For any $\gamma \in [-1/2, 1/2]$, PPT algorithm D, $\lambda, n \in \mathbb{N}$, $\mathsf{params} = (\mathsf{pp}, \mathsf{msk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, and message $m \in \mathcal{M}_\lambda$, we say D is γ-$\mathsf{Dist}_{\mathsf{params}}^{\mathsf{nrml},0}$ for m if

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \begin{array}{c} b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \geq \frac{1}{2} + \gamma,$$

where the probability is taken over the random coins used during encryption, the random coins of D, and the choice of b.

Similarly, for any $i \in \{0, \ldots, n-1\}$ we can define D to be γ-$\mathsf{Dist}^{i,i+1}_{\mathsf{params}}$ for m if

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\}; \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, i+b) \end{array}\right] \geq \frac{1}{2} + \gamma.$$

Finally, we also define D to be γ-$\mathsf{Dist}^{n}_{\mathsf{params}}$ for messages m_0, m_1 if

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\}; \\ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m_b, n) \end{array}\right] \geq \frac{1}{2} + \gamma.$$

Definition 5.1.4 (decoder-based normal hiding security)**.** A PLBE scheme is said to satisfy decoder-based normal hiding security if for any PPT adversary $\mathcal{A}$, nonnegligible function $\gamma(\cdot)$, and polynomial $q(\cdot)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\gamma(\lambda) > 1/q(\lambda)$, $p^{\mathsf{dec,nrml}}_{\mathcal{A},\gamma,q}(\lambda) \leq \mathrm{negl}(\lambda)$, where $p^{\mathsf{dec,nrml}}_{\mathcal{A},\gamma,q}(\lambda)$ is defined as follows:

$$\Pr\left[D \text{ is } \gamma(\lambda)\text{-}\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}} \text{ for } m : \ \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \mathsf{params} = \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ (D, m) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \end{array}\right].$$

Definition 5.1.5 (decoder-based index hiding security)**.** A PLBE scheme is said to satisfy decoder-based index hiding security if for any PPT adversary $\mathcal{A}$, nonnegligible function $\gamma(\cdot)$, and polynomial $q(\cdot)$ there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\gamma(\lambda) > 1/q(\lambda)$ and $i^* \in \{0, \ldots, n-1\}$, $p^{\mathsf{dec,ind}}_{\mathcal{A},\gamma,q}(\lambda, i^*) \leq \mathrm{negl}(\lambda)$, where $p^{\mathsf{dec,ind}}_{\mathcal{A},\gamma,q}(\lambda, i^*)$ is defined as follows:

$$\Pr\left[D \text{ is } \gamma(\lambda)\text{-}\mathsf{Dist}^{i^*,i^*+1}_{\mathsf{params}} \text{ for } m : \ \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \mathsf{params} = \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ (D, m) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in\{1,\ldots,n\}\setminus\{i^*+1\}}\right) \end{array}\right].$$

Definition 5.1.6 (decoder-based message hiding security)**.** A PLBE scheme is said to satisfy decoder-based message hiding security if for any PPT adversary

$\mathcal{A}$, nonnegligible function $\gamma(\cdot)$, and polynomial $q(\cdot)$ there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\gamma(\lambda) > 1/q(\lambda)$, $p^{\mathsf{dec,msg}}_{\mathcal{A},\gamma,q}(\lambda) \leq \mathrm{negl}(\lambda)$, where $p^{\mathsf{dec,msg}}_{\mathcal{A},\gamma,q}(\lambda)$ is defined as follows:

$$\Pr\left[\begin{array}{l} D \text{ is } \gamma(\lambda)\text{-}\mathsf{Dist}^n_{\mathsf{params}} \text{ for } m_0, m_1 : \end{array} \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda); \\ \mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \\ \mathsf{params} = \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\in[n]}\right); \\ (D, m_0, m_1) \leftarrow \mathcal{A}\left(\mathsf{pp}, \{\mathsf{sk}_i\}_{i\in[n]}\right) \end{array}\right].$$

5.2 Traitor tracing from decoder-based PLBE

Consider a PLBE scheme $\mathsf{PLBE} = (\mathsf{PLBE.Setup}, \mathsf{PLBE.Enc}, \mathsf{PLBE.Enc\text{-}index}, \mathsf{PLBE.Dec})$ with decoder-based security. We will use PLBE to construct a TT scheme $\mathcal{T} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ as follows. The construction is identical to the transformation in [25]; however, the security proof provided in [25] was not correct. Concretely, to argue correctness of tracing they incorrectly leveraged the indistinguishability-based security of the underlying PLBE scheme. We show that the same transformation could be proven to satisfy correct tracing if one starts with a PLBE scheme that achieves decoder-based security.

$\mathsf{Setup}(1^\lambda, 1^n)$: The setup algorithm samples parameters as $(\mathsf{pp}, \mathsf{msk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$ $\leftarrow \mathsf{PLBE.Setup}(1^\lambda, 1^n)$. The public parameters are pp, the master secret key is msk, and the n secret keys are $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.

$\mathsf{Enc}(\mathsf{pp}, m)$: The encryption algorithm outputs $\mathsf{ct} \leftarrow \mathsf{PLBE.Enc}(\mathsf{pp}, m)$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: The decryption algorithm outputs $\mathsf{PLBE.Dec}(\mathsf{sk}, \mathsf{ct})$.

$\mathsf{Trace}^D(\mathsf{msk}, 1^y, m_0, m_1)$: Let $\epsilon = 1/y$ and $W = \lambda \cdot (n \cdot y)^2$. For $i = 0$ to n, the trace algorithm does the following:

1. It first sets $\mathsf{count}_i = 0$. For $j = 1$ to W, it does the following:
   (a) It chooses a bit $b_{i,j} \leftarrow \{0, 1\}$ and sets $\mathsf{ct}_{i,j} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m_b, i)$. If $D(\mathsf{ct}_{i,j}) = b_{i,j}$, it sets $\mathsf{count}_i = \mathsf{count}_i + 1$.
2. It sets $\hat{p}_i = \mathsf{count}_i / W$.

The trace algorithm outputs every index $i \in \{1, 2, \ldots, n\}$ such that $\hat{p}_{i-1} - \hat{p}_i \geq \epsilon/4n$.

Correctness This follows directly from the first correctness property of the PLBE scheme.

5.2.1 IND-CPA security

We would like to point out that the scheme is IND-CPA secure even if PLBE only satisfies 0-query security. In other words, we do not need PLBE to achieve stronger decoder-based security. Thus, the proof of IND-CPA security is identical to that provided in [25]. Below we provide a high level sketch.

Theorem 5.2.1. *Assuming the PLBE scheme* $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ *satisfies the security properties in Definitions 5.1.4, 5.1.5 and 5.1.6, the traitor tracing scheme described above is IND-CPA secure* (Definition 4.1.1).

Proof. We will construct a sequence of $2n + 2$ hybrid experiments to prove

IND-CPA security. The first experiment, Hybrid H_0, is exactly the IND-CPA game.

Hybrid H_0 In this experiment, the challenger sends public parameters pp, receives m_0, m_1 from $\mathcal{A}$, and sends $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_0)$ to $\mathcal{A}$.

Hybrid $H_{i,b}$ (for $i \leq n, b \in \{0,1\}$) This experiment is identical to the IND-CPA experiment, except that the adversary, after sending challenge messages m_0, m_1, receives $\mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, i, m_b)$.

Hybrid H_1 In this experiment, the challenger sends public parameters pp, receives m_0, m_1 from $\mathcal{A}$, and sends $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_1)$ to $\mathcal{A}$.

For any PPT adversary $\mathcal{A}$, let $p_{\mathcal{A},x}(\cdot)$ be a function of λ that denotes the probability of $\mathcal{A}$ outputting 0 in H_x. Note that $p_{\mathcal{A},0} - p_{\mathcal{A},1}$ is the advantage of $\mathcal{A}$ in the IND-CPA security game.

Claim 5.2.1. *Assuming* PLBE *satisfies* Definition 5.1.4*, for any PPT adversary $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$ and $b \in \{0,1\}$, $|p_{\mathcal{A},b} - p_{\mathcal{A},0,b}| \leq negl(\lambda)$.*

This follows from decoder-based indistinguishability of normal and 0-index encryptions (Definition 5.1.4) of PLBE.

Claim 5.2.2. *Assuming* PLBE *satisfies* Definition 5.1.5*, for any PPT adversary $\mathcal{A}$, there exists a negligible function such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$,*

and $i \in [n]$, $p_{\mathcal{A},i-1,b} - p_{\mathcal{A},i,b} \leq negl(\lambda)$.

This follows from the decoder-based index hiding security notion (Definition 5.1.5) of PLBE.

Claim 5.2.3. *Assuming* PLBE *satisfies* Definition 5.1.6*, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function such that for all* $\lambda \in \mathbb{N}$, $p_{\mathcal{A},n,0} - p_{\mathcal{A},n,1} \leq negl(\lambda)$.

This follows from the decoder-based message hiding security notion (Definition 5.1.6) of PLBE.

From the above claims, it follows that $p_{\mathcal{A},0} - p_{\mathcal{A},1}$ is bounded by a negligible function. ■

5.2.2 Correctness of tracing

Next, we will show that the false trace probability is bounded by a negligible function, and the correct trace probability is close to the probability of $\mathcal{A}$ outputting an ϵ-successful decoding box.

First, we will introduce some notation. Given any pirate decoder box D and messages m_0, m_1, for any $i \in \{0, 1, \ldots, n\}$, let

$$p_i^D = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, i, m_b)],$$

where the probability is taken over random coins of decoder D as well as the randomness used during encryption. Similarly, let $p_{\mathsf{nrml}}^D = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, m_b)]$.

False trace probability First, we show that the tracing algorithm never falsely accuses any user. Formally, we prove the following.

Theorem 5.2.2. *For every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$, and nonnegligible function $\epsilon(\cdot)$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$,*

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq negl(\lambda),$$

where $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ *is as defined in* Definition 4.1.2.

Proof. We will skip the dependence of $\epsilon(\cdot)$ on λ for simplicity of notation. Let S be the set of keys queried and D the decoder output by $\mathcal{A}$. For $i \in \{1, 2, \ldots, n\}$, we define events $\mathsf{Diff\text{-}Adv}_i^D : p_{i-1}^D - p_i^D > \epsilon/8n$ and $\mathsf{Diff\text{-}Adv}^D : \bigvee_{k \in \{1,\ldots,n\} \setminus S} \mathsf{Diff\text{-}Adv}_k^D$.

First, note that the probability of the event *false trace* can be rewritten as follows by conditioning on the events defined above:

$$\Pr[\mathsf{Fal\text{-}Tr}] \leq \Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}^D}] + \sum_{i \in \{1,\ldots,n\}} \Pr[i \notin S \wedge \mathsf{Diff\text{-}Adv}_i^D].$$

We will show that each of these terms is bounded by a negligible function.

Lemma 5.2.1. *For every PPT adversary $\mathcal{A}$ there exists a negligible function $negl_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}^D}] \leq negl_1(\lambda).$$

Proof. The proof of this lemma follows from Chernoff bounds. Let n be the number of users chosen by the adversary $\mathcal{A}$. Fix any $i \in \{1, \ldots, n\} \setminus S$ and decode box D. Let us consider the probability that the output of Trace algorithm includes i, given that $\mathsf{Diff\text{-}Adv}_i^D$ does not occur. Note that the tracing algorithm includes i in the traitor set if the estimates $\hat{p}_{i-1}$ and $\hat{p}_i$ differ by at least $\epsilon/4n$.

Let $X_{k,j}$ denote the random variable that is 1 if $D(\mathsf{ct}_{k,j}) = b_{k,j}$ for $k \in \{i-1, i\}$ and $j \in \{1, 2, \ldots, W\}$ (here the randomness is over the choice of $b_{k,j}$ and the randomness used by $\mathsf{Enc\text{-}index}$ and D) and $Z_{i,j} = X_{i-1,j} - X_{i,j}$. Then $(\sum_{j=1}^{W} X_{k,j})/W = \hat{p}_k$ and $\mu_i = \mathbb{E}[Z_{i,j}] = p_{i-1}^D - p_i^D$.

Since the Z_i's are independent samples, using Chernoff bounds, we get that $\Pr[\sum_j Z_j/W > 2\mu_i] \leq 2^{-O(\lambda)}$. Using this, we can write that for every $i \in \{1, \ldots, n\} \setminus S$, $\Pr[\mathsf{Fal\text{-}Tr} \wedge i \in T \mid \overline{\mathsf{Diff\text{-}Adv}^D}] \leq 2^{-O(\lambda)}$, where T denotes the set of indices output by Trace. Finally, using a union bound, we get that

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}^D}] \leq n \cdot 2^{-O(\lambda)} = \mathrm{negl}_1(\lambda).$$

■

Lemma 5.2.2. *Assuming* PLBE *is a secure PLBE scheme satisfying the decoder-based index hiding security property* (Definition 5.1.5)*, for every PPT adversary* $\mathcal{A}$*, polynomial* $q(\cdot)$*, and nonnegligible function* $\epsilon(\cdot)$*, there exists a negligible function* $negl_2(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *satisfying* $\epsilon(\lambda) > 1/q(\lambda)$ *and* $i \in \{1, 2, \ldots, n\}$*,*

$$\Pr[i \notin S \wedge \mathsf{Diff\text{-}Adv}_i^D] \leq negl_2(\lambda),$$

where n is the number of users, S is the set of key queries, and D is the decoder box sent by $\mathcal{A}$.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$, and nonnegligible functions $\epsilon(\cdot), \delta(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, there exists an $i^* \in \{1, 2, \ldots, n\}$ such that $\Pr[i^* \notin S \wedge \mathsf{Diff\text{-}Adv}_{i^*}^{D}] \geq \delta(\lambda)$. Then we can use $\mathcal{A}$ to build a PPT reduction algorithm $\mathcal{B}$ that breaks the index hiding security property of PLBE.

The reduction algorithm $\mathcal{B}$ first receives 1^n from the adversary, which it forwards to the challenger. It then receives the PLBE public parameters pp from the challenger, which it sends to $\mathcal{A}$. Next, it chooses an index $i \leftarrow \{1, 2, \ldots, n\}$ and sends it to the PLBE challenger.[1] It receives secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{i-1}, \mathsf{sk}_{i+1}, \ldots, \mathsf{sk}_n$. The adversary $\mathcal{A}$ queries for secret keys. If $\mathcal{A}$ queries for i, $\mathcal{B}$ sends an empty decoding box to the PLBE challenger. Otherwise, on receiving query $j \neq i$ from $\mathcal{A}$, it sends sk_j to $\mathcal{A}$. After all queries, the adversary sends a decoding box D and messages m_0, m_1 to $\mathcal{B}$. The reduction algorithm chooses a uniformly random bit $b' \leftarrow \{0, 1\}$ and sends $D, m_{b'}$ to the PLBE challenger.

Let $p_{j,b}^{D} = \Pr[D(\mathsf{ct}) = b \ : \ \mathsf{ct} \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, j, m_b)]$, where the probability is taken over the coins of decoder D and the encryption algorithm. Recall that we have $\Pr[i^* \notin S \wedge \mathsf{Diff\text{-}Adv}_{i^*}] \geq \delta(\lambda)$. Therefore, we can write

[1] In other words, the reduction algorithm randomly guesses the index hiding challenger with which it interacts.

that

$$\Pr\left[i^* \notin S \wedge \left((p^D_{i^*-1,0} + p^D_{i^*-1,1})/2 - (p^D_{i^*,0} + p^D_{i^*,1})/2\right) \geq \epsilon/8n\right] \geq \delta(\lambda)$$
$$\Rightarrow \Pr\left[i = i^* \wedge i^* \notin S \wedge \left((p^D_{i-1,0} + p^D_{i-1,1})/2 - (p^D_{i,0} + p^D_{i,1})/2\right) \geq \epsilon/8n\right] \geq \delta(\lambda)/n.$$

Thus, we can also write that there exists a bit b such that

$$\Pr\left[i = i^* \wedge i^* \notin S \wedge \left(p^D_{i-1,b} - p^D_{i,b}\right) \geq \epsilon/8n\right] \geq \delta(\lambda)/n.$$

Now since the reduction algorithm $\mathcal{B}$ simply randomly guesses this bit b, thus we have that with probability at least $\delta/2n$, $\mathcal{B}$ outputs a decoding box D and a message m_b such that D can distinguish between encryptions of m_b to indices $i-1$ and i with advantage at least $\epsilon/8n$. Thus, the lemma follows.

■

From the above lemmas, it follows that the probability of false trace is at most $\text{negl}_1(\lambda) + n \cdot \text{negl}_2(\lambda)$, and thus the theorem follows.

■

Correct trace probability Now we show that whenever the adversary outputs a good decoder, then with all but negligible probability the tracing algorithm outputs a nonempty set T. Combining this with Theorem 5.2.2, we get that the tracing algorithm correctly traces. Formally, we show the following.

Theorem 5.2.3. *For every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$, and nonnegligible function $\epsilon(\cdot)$, there exists a negligible function $\text{negl}(\cdot)$ such that for all*

$\lambda \in \mathbb{N}$ *satisfying* $\epsilon(\lambda) > 1/q(\lambda)$,

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathit{negl}(\lambda),$$

where $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ *and* $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\cdot)$ *are as defined in* Definition 4.1.2.

Proof. Let us start by analyzing the probability that the tracing algorithm outputs a nonempty set T. First, we know that if event $\mathsf{Good\text{-}Decoder}$ occurs, then $p^D_{\mathsf{nrml}} \geq 1/2 + \epsilon$. Next, let S be the set of indices $i \in \{1, \ldots, n\}$ such that $p^D_{i-1} - p^D_i > \epsilon/2n$. Using Chernoff bounds, we get that

$$\forall\ i \in S, \quad \Pr\left[\hat{p}^D_{i-1} - \hat{p}^D_i < \epsilon/4n\right] \leq 2^{-O(\lambda)} = \mathrm{negl}_1(\lambda). \tag{5.1}$$

Note that by message hiding security of the underlying PLBE scheme, we have that $p^D_n \leq 1/2 + \mathrm{negl}_2(\lambda)$ for some negligible function $\mathrm{negl}_2(\cdot)$. Also, by indistinguishability of *normal* and index 0 ciphertexts, we have that $p^D_{\mathsf{nrml}} - p^D_0 \leq \mathrm{negl}_3(\lambda)$ for some negligible function $\mathrm{negl}_3(\cdot)$. Thus, $p^D_0 - p^D_n \geq \epsilon - \mathrm{negl}_2(\lambda) - \mathrm{negl}_3(\lambda) > \epsilon/2$. Given this, we can conclude that the set S as defined above (i.e., for $i \in S$, $p^D_{i-1} - p^D_i > \epsilon/2n$) must be nonempty whenever event $\mathsf{Good\text{-}Decoder}$ occurs. Combining this with Eq. (8.18), we get that

$$\Pr[T \neq \emptyset] \geq (1 - \mathrm{negl}_1(\lambda)) \cdot \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda).$$

Finally, combining with Theorem 5.2.2, we get that

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda).$$

This concludes the proof. ∎

5.3 Decoder-based PLBE from 1-query secure PLBE

Let $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ be a PLBE scheme that satisfies 1-query security. We will show that the same scheme also satisfies decoder-based security.

Lemma 5.3.1. *If* PLBE *satisfies* 1*-query normal hiding security* (Definition 5.1.1)*, then it also satisfies decoder-based normal hiding security* (Definition 5.1.4)*.*

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, nonnegligible function $\gamma(\cdot)$, polynomials $q(\cdot), r(\cdot)$, and an infinite sequence of security parameters $\Lambda = \{\lambda_i\}_{i\in\mathbb{N}}$ such that for all $\lambda \in \Lambda$, $\gamma(\lambda) > 1/q(\lambda)$ and $p^{\mathsf{dec},\mathsf{nrml}}_{\mathcal{A},\gamma,q}(\lambda) \geq 1/r(\lambda)$. We will use $\mathcal{A}$ that plays the 1-query normal hiding security game to build a PPT reduction algorithm $\mathcal{B}$ that plays the decoder-based normal hiding security game as follows.

For any $\lambda \in \mathbb{N}$, the reduction algorithm first receives 1^n from $\mathcal{A}$, which it forwards to the challenger. It then receives pp and n secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ from the challenger, which it forwards to $\mathcal{A}$. The adversary $\mathcal{A}$ outputs D and m. The reduction algorithm then queries the challenger for an encryption of m for index 0 (recall that the reduction algorithm is allowed one query). Let the challenger's response be ct_1. It then computes $\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m)$. Next, it sends challenge message m and receives ct^*, which is either a normal encryption of m or an encryption of m for index 0. The reduction algorithm chooses a random bit $\beta \leftarrow \{0,1\}$ and checks if $D(\mathsf{ct}_\beta) = D(\mathsf{ct}^*)$. If so, it outputs $b' = \beta$; otherwise it outputs $b' = 1 - \beta$ as its guess.

Let us now analyze $\mathcal{B}$'s advantage. We need to show that there exist polynomials $q_{\mathcal{B}}(\cdot)$ and an infinite sequence $\Lambda_{\mathcal{B}} = \{\lambda_i\}_i$ such that for all $\lambda \in \Lambda_{\mathcal{B}}$, $p_{\mathcal{B}}^{1,\mathsf{nrml}} \geq 1/2 + 1/q_{\mathcal{B}}(\lambda)$. Let $\Lambda_B = \Lambda$ and $q_{\mathcal{B}}(\cdot) = q^2(\cdot) \cdot r(\cdot)/2$. Fix any $\lambda \in \Lambda$, and let $\gamma = \gamma(\lambda)$, $q = q(\lambda)$, $r = r(\lambda)$. Let b denote the 1-query challenger's choice (recall the challenger chooses $b \leftarrow \{0,1\}$; if $b = 0$, it sends a normal encryption, else it sends an encryption to index 0). First, let us fix $\mathsf{params} = (\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \leq n}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and $(D, m) \leftarrow \mathcal{A}\big(\mathsf{pp}, \{\mathsf{sk}_i\}_{i \leq n}\big)$ such that

$$\Pr\left[D(\mathsf{ct}_b) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] = \frac{1}{2} + \alpha_{\mathsf{params},D,m}$$

for some $\alpha_{\mathsf{params},D,m} \in [-1/2, 1/2]$. Next, consider the following probability:

$$\rho_{\mathsf{params},D,m} = \Pr\left[b = b' : \begin{array}{l} b \leftarrow \{0,1\}; \beta \leftarrow \{0,1\}; \\ \mathsf{ct}_0^* \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \mathsf{ct}_1^* \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0); \\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0); \\ b' = \beta \text{ if } D(\mathsf{ct}_b^*) = D(\mathsf{ct}_\beta), \text{ else } b' = 1 - \beta \end{array}\right].$$

Since the decoder D is run on ciphertexts $\mathsf{ct}_b^*, \mathsf{ct}_\beta$ independently, we could rewrite the above probability as follows:

$$\begin{aligned} \rho_{\mathsf{params},D,m} = &\Pr\left[D(\mathsf{ct}_b^*) = b : \begin{array}{c} b \leftarrow \{0,1\}; \mathsf{ct}_0^* \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1^* \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \\ &\times \Pr\left[D(\mathsf{ct}_\beta) = \beta : \begin{array}{c} \beta \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \\ &+ \Pr\left[D(\mathsf{ct}_b^*) \neq b : \begin{array}{c} b \leftarrow \{0,1\}; \mathsf{ct}_0^* \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1^* \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \\ &\times \Pr\left[D(\mathsf{ct}_\beta) \neq \beta : \begin{array}{c} \beta \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right] \\ = &\Pr\left[D(\mathsf{ct}_b) = b : \begin{array}{c} b \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right]^2 \end{aligned}$$

$$
\begin{aligned}
&+ \Pr\left[D(\mathsf{ct}_b) \neq b : \begin{array}{l} b \leftarrow \{0,1\}; \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, m); \\ \mathsf{ct}_1 \leftarrow \mathsf{Enc\text{-}index}(\mathsf{msk}, m, 0) \end{array}\right]^2 \\
&= \left(\frac{1}{2} + \alpha_{\mathsf{params},D,m}\right)^2 + \left(\frac{1}{2} - \alpha_{\mathsf{params},D,m}\right)^2 \\
&= \frac{1}{2} + 2 \cdot \alpha^2_{\mathsf{params},D,m}.
\end{aligned}
$$

Thus, we get that for any decoder D that is δ-$\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}}$ for message m,

$$
\rho_{\mathsf{params},D,m} = 1/2 + 2 \cdot \alpha^2_{\mathsf{params},D,m} \geq 1/2 + 2 \cdot \delta^2.
$$

Also, since $\alpha^2_{\mathsf{params},D,m} \geq 0$, we get that for every decoder D, $\rho_{\mathsf{params},D,m} \geq 1/2$. Therefore, since the adversary $\mathcal{A}$ outputs a $1/q$-$\mathsf{Dist}^{\mathsf{nrml},0}_{\mathsf{params}}$ box with probability at least $1/r$, we get that the reduction algorithm $\mathcal{B}$'s winning probability $p^{1,\mathsf{nrml}}_{\mathcal{B},n}$ (as defined in Definition 5.1.1) is

$$
\begin{aligned}
p^{1,\mathsf{nrml}}_{\mathcal{B},n} &\geq \frac{1}{r} \cdot \left(\frac{1}{2} + \frac{2}{q^2}\right) + \left(1 - \frac{1}{r}\right) \cdot \left(\frac{1}{2}\right). \\
&\geq \frac{1}{2} + \frac{2}{r \cdot q^2}.
\end{aligned}
$$

This concludes our proof. ■

Lemma 5.3.2. *If* PLBE *satisfies* 1*-query index hiding security* (Definition 5.1.2)*, then it also satisfies decoder-based index hiding security* (Definition 5.1.5)*.*

The proof of this lemma is identical to the proof of Lemma 5.3.1, except that the reduction algorithm queries for either a special encryption of m for index i or a special encryption of m for index $i+1$ (depending on $\beta \leftarrow \{0,1\}$).

Lemma 5.3.3. *If* PLBE *satisfies* 1*-query message hiding security* (Definition 5.1.3)*, then it also satisfies decoder-based message hiding security* (Definition 5.1.6)*.*

The proof of this lemma is identical to the proof of Lemma 5.3.1, except that the reduction algorithm queries for either a special encryption of m_0 for index n or a special encryption of m_1 for index n (depending on $\beta \leftarrow \{0,1\}$).

5.4 Constructing PLBE from Mixed FE and ABE

In this section, we construct a private linear broadcast encryption (PLBE) scheme from any key-policy attribute-based encryption (KP-ABE) scheme and a Mixed FE scheme. Our construction inherits the message space of the underlying KP-ABE scheme. Also, we show that if the underlying ABE scheme is selectively secure, and the mixed FE scheme satisfies 1-query restricted function and accept indistinguishability properties, then our PLBE scheme satisfies 1-query normal, index, and message hiding security properties.

Outline The idea is to use the ABE system to encrypt a message with attributes being set to either the "normal" ciphertext (i.e., encryption of the canonical always-accepting function) or a special (secret key) ciphertext which encrypts the comparison function depending on the type of PLBE encryption operation being performed. Each user's secret key will be an ABE private key. Here the ABE private key is generated for the Mixed FE decryption circuit in which a Mixed FE secret key, corresponding to the user's index, is hardwired. The high level intuition is that when the attribute is a normal functional encryption ciphertext, then all keys decrypt it to 1; thus any user with an appropriate ABE key could perform the decryption. Also, when the

attribute is set to be a special ciphertext (that encrypts comparison with some index i), then only those users whose indices are larger than the threshold i set can perform the decryption. For proving security, we rely on the fact that special ciphertexts are indistinguishable to any adversary that does not have distinguishing secret keys. Below we provide a detailed overview.

The PLBE setup algorithm starts by sampling an ABE key pair $(\mathsf{abe.pp}, \mathsf{abe.msk})$ and a mixed FE key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$. To generate the private key for the ith user, it first generates a mixed FE secret key $\mathsf{mixed.sk}_i$ for message i, and later computes an ABE key $\mathsf{abe.sk}_i$ for predicate $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, i.e., $\mathsf{Mixed\text{-}FE}$ decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired. For PLBE normal encryption, one simply computes ciphertext ct as an encryption of message m under attributes $\mathsf{ct}_{\mathsf{attr}}$, where $\mathsf{ct}_{\mathsf{attr}}$ is a mixed FE normal ciphertext. For encrypting a message to index i, the encryption algorithm works identically, except now the attribute is set to be a special ciphertext corresponding to function *greater than* i. Finally, the PLBE decryption is simply the ABE decryption algorithm.

Now correctness follows directly from the correctness of ABE and functional encryption schemes. For proving security, the main idea is as follows: Suppose there exists an adversary that can distinguish between PLBE normal ciphertexts and index 0 ciphertexts; then it can be used to distinguish between mixed FE normal ciphertexts and secret key ciphertexts encrypting function *greater than* 0 (note that this is an always-accepting function). In other words, such an attack can be used to break the restricted accept indistinguishability

property of the mixed FE scheme. Similarly, we can also reduce a successful attack on the index hiding, or message hiding security to an attacker on restricted function indistinguishability of mixed FE or ABE security, respectively. Below we describe our construction $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ for message spaces $\{\mathcal{M}_\kappa\}_\kappa$ in detail.

The following construction only achieves the statistical notion of correctness. In Chapter A, we provide an alternate construction that achieves perfect correctness from the same assumptions.

5.4.1 Construction

Let $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ be a KP-ABE scheme for a set of attribute spaces $\{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\{\mathcal{C}_\kappa\}_\kappa$, and message spaces $\{\mathcal{M}_\kappa\}_\kappa$, and let $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ be a mixed FE scheme for function classes $\{\mathcal{F}_\kappa\}_\kappa$ and message space $\{\mathcal{I}_\kappa\}_\kappa$, with ciphertexts of length $\ell(\lambda, \kappa)$. For every n, let $\kappa = \kappa(n)$ be the lexicographically smallest functionality index such that every string of length $\log(n)$ can be uniquely represented in message space $\mathcal{I}_\kappa$ (i.e., $\{0,1\}^{\log(n)} \subseteq \mathcal{I}_\kappa$), and function class $\mathcal{F}_\kappa$ contains the "comparison" ($>$) operator. Also, let $\widetilde{\kappa} = \widetilde{\kappa}(\lambda, \kappa)$ be the lexicographically smallest functionality index such that every string of length $\ell(\lambda, \kappa)$ can be uniquely represented in attribute class $\mathcal{X}_{\widetilde{\kappa}}$ (i.e., $\{0,1\}^{\ell(\lambda,\kappa)} \subseteq \mathcal{X}_{\widetilde{\kappa}}$), and $\mathcal{C}_{\widetilde{\kappa}}$ contains mixed FE decryption circuit corresponding to functionality index κ. Below we describe our construction.

- $\mathsf{Setup}(1^\lambda, 1^n) \to \left(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i \le n}\right)$. The setup algorithm runs $\mathsf{ABE.Setup}$

and $\mathsf{Mixed.Setup}$ to generate ABE and mixed FE public parameters and master secret key as $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\tilde{\kappa}})$ and $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$. Next, it runs $\mathsf{Mixed.KeyGen}$ to generate n mixed FE secret keys $\mathsf{mixed.sk}_i$ as

$$\forall\ i \leq n, \quad \mathsf{mixed.sk}_i \leftarrow \mathsf{Mixed.KeyGen}(\mathsf{mixed.msk}, i).$$

Let $C_{\mathsf{mixed.sk}_i}$ denote the circuit $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, i.e., Mixed-FE decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired. Next, it computes n ABE secret keys $\mathsf{abe.sk}_i$ as

$$\forall\ i \leq n, \quad \mathsf{abe.sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_i}).$$

Finally, it sets $\mathsf{pp} = (\mathsf{abe.pp}, \mathsf{mixed.pp})$, $\mathsf{msk} = (\mathsf{abe.msk}, \mathsf{mixed.msk})$ and $\mathsf{sk}_i = \mathsf{abe.sk}_i$ for $i \leq n$.

- $\mathsf{Enc}(\mathsf{pp}, m) \rightarrow \mathsf{ct}$. Let $\mathsf{pp} = (\mathsf{abe.pp}, \mathsf{mixed.pp})$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$. Next, it encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, and outputs ciphertext ct.

- $\mathsf{Enc\text{-}index}(\mathsf{msk}, m, i) \rightarrow \mathsf{ct}$. Let $\mathsf{msk} = (\mathsf{abe.msk}, \mathsf{mixed.msk})$ and let comp_i denote the comparison function $\overset{?}{>} i$, i.e., $\mathsf{comp}_i(x) = 1$ iff $x > i$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_i)$. Next, it encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$ and outputs ciphertext ct.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$ or $\perp$. The decryption algorithm runs $\mathsf{ABE.Dec}$ on ct using key sk as $y = \mathsf{ABE.Dec}(\mathsf{sk}, \mathsf{ct})$ and sets y as the output of decryption.

5.4.2 Correctness

For all $\lambda, n \in \mathbb{N}$, message $m \in \mathcal{M}_\lambda$, public parameters and master secret keys $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$, $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$, the secret keys sk_i for $i \leq n$ are simply the ABE keys $\mathsf{abe.sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_i})$. For any index $i \leq n$, consider the following two cases:

1. **Normal encryption.** For any ciphertext computed as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, where $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$, we know that with all but negligible probability $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = 1$ by correctness of the mixed FE scheme. In other words, $C_{\mathsf{mixed.sk}_i}(\mathsf{ct}_{\mathsf{attr}}) = 1$. Therefore, by correctness of the ABE scheme, we get that with all but negligible probability $\mathsf{ABE.Dec}(\mathsf{abe.sk}_i, \mathsf{ct}) = m$.

2. **Index encryption.** For any index $0 \leq j \leq n$ and ciphertext ct computed as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$, where $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_j)$, we know that with all but negligible probability
$$\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = \begin{cases} 1 \text{ if } i > j, \\ 0 \text{ otherwise} \end{cases}$$
by correctness of the mixed FE scheme. In other words, $C_{\mathsf{mixed.sk}_i}(\mathsf{ct}_{\mathsf{attr}}) = \mathsf{comp}_j(i) = (i > j)$. Therefore, by correctness of the ABE scheme, we

have that with all but negligible probability $\mathsf{ABE.Dec}(\mathsf{abe.sk}_i, \mathsf{ct}) = m$ for $i > j$ and $\perp$ otherwise.

Therefore, PLBE satisfies the PLBE correctness condition.

5.4.3 Security

We will now show that the scheme described above is 1-query secure per Definitions Definition 5.1.1, Definition 5.1.2, and Definition 5.1.3. In other words, it satisfies normal hiding, index hiding, and message hiding security properties. Formally, we prove the following.

Theorem 5.4.1. *If* $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ *is a selectively secure ABE scheme for a set of attribute spaces* $\{\mathcal{X}_\kappa\}_\kappa$*, predicate classes* $\{\mathcal{C}_\kappa\}_\kappa$*, and message spaces* $\{\mathcal{M}_\kappa\}_\kappa$ *satisfying* Definition 3.5.2*, and* $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ *is a mixed FE scheme, for function classes* $\{\mathcal{F}_\kappa\}_\kappa$ *and message spaces* $\{\mathcal{I}_\kappa\}_\kappa$*, satisfying* 1*-query restricted function indistinguishability* (Definition 4.2.2) *and* 1*-query restricted accept indistinguishability* (Definition 4.2.4) *properties, then* $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ *is a secure PLBE scheme, for messages spaces* $\{\mathcal{M}_\kappa\}_\kappa$*, satisfying* 1*-query normal, index, and message hiding security properties per Definitions* Definition 5.1.1, Definition 5.1.2*, and* Definition 5.1.3*, respectively.*

Our proof is divided into three components/lemmas, one for each PLBE security property. Let $\mathcal{A}$ be any PPT adversary that wins the normal/index/message

hiding game with nonnegligible advantage. We argue that such an adversary must break the security of at least one underlying primitive.

Normal hiding security

Lemma 5.4.1. *If* Mixed-FE = (Mixed.Setup, Mixed.Enc, Mixed.SK-Enc, Mixed.Dec, Mixed.KeyGen) *is a mixed FE scheme satisfying the* 1*-query restricted accept indistinguishability* (Definition 4.2.4) *property, then* PLBE = (Setup, Enc, Dec, Enc-index) *is a PLBE scheme satisfying the* 1*-query normal hiding security property per* Definition 5.1.1.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $\mathcal{A}$'s advantage in a 1-query normal hiding security game is nonnegligible. We construct an algorithm $\mathcal{B}$ that can distinguish normal encryptions from secret key encryptions, therefore breaking the 1-query restricted accept indistinguishability security of the mixed FE scheme.

The reduction algorithm $\mathcal{B}$ receives 1^n from $\mathcal{A}$. It sets $\kappa, \widetilde{\kappa}$ as in the construction and sends κ as the functionality index and comp_0 (i.e., comparison with 0) as its challenge function to the mixed FE challenger. The challenger generates a key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$ and sends $\mathsf{mixed.pp}$ as the public parameters and challenge ciphertext $\mathsf{ct}^*_{\mathsf{attr}}$ to $\mathcal{B}$. Next, $\mathcal{B}$ makes an encryption query for function comp_0. Let the challenger's response be ciphertext $\mathsf{ct}_{\mathsf{attr}}$. $\mathcal{B}$ then queries the challenger on n messages i $(\leq n)$ for corresponding mixed FE secret keys and receives back keys $\mathsf{mixed.sk}_i$ for $i \leq n$. It then chooses an ABE key pair $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$ and

computes n ABE keys as $\mathsf{abe.sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_i})$. Next, it sends $(\mathsf{abe.pp}, \mathsf{mixed.pp})$ and $\{\mathsf{abe.sk}_i\}_{i \leq n}$ as the PLBE public parameters and secret keys to $\mathcal{A}$. After receiving all the keys, $\mathcal{A}$ sends its challenge message m^* to $\mathcal{B}$, and it can also make a single encryption query for message m on index 0. Here $\mathcal{A}$ is allowed to make the encryption query either before or after challenge query. The reduction algorithm $\mathcal{B}$ responds to each query as follows: $\mathcal{B}$ encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$ and sends ct to $\mathcal{A}$ as its response to the encryption query. Also, it computes ciphertext ct^* as $\mathsf{ct}^* \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}^*_{\mathsf{attr}}, m^*)$ and sends ct^* as the challenge ciphertext to $\mathcal{A}$. Note that $\mathcal{A}$ could instead have sent its challenge query before sending the index encryption query. Also, $\mathcal{B}$ does not need to query the mixed FE challenger for answering any query at this point as it already has ciphertexts $\mathsf{ct}_{\mathsf{attr}}, \mathsf{ct}^*_{\mathsf{attr}}$. Finally, $\mathcal{A}$ sends its guess b to $\mathcal{B}$, and $\mathcal{B}$ forwards b as its own guess.

First, note that both $\mathcal{A}$ and $\mathcal{B}$ are allowed to make at most 1 index encryption and polynomially many secret key encryption queries, respectively. Also, note that $\mathcal{B}$ sends its secret key encryption query as well as its challenge query before making making key generation queries; thus $\mathcal{B}$ is an admissible adversary in the 1-query restricted accept indistinguishability game. Since $\mathcal{A}$ is only allowed to make encryption queries to index 0 (in the 1-query normal hiding security game), thus $\mathcal{B}$ queries the mixed FE challenger on functions comp_0 which are always accepting functions and therefore admissible queries per the restricted accept indistinguishability game. Next, for each query made

by $\mathcal{A}$, $\mathcal{B}$ queries the mixed FE challenger exactly once, and thus all the queries are honestly and exactly answered. Finally, note that if the mixed FE challenger computed $\mathsf{ct}^*_{\mathsf{attr}}$ as a normal functional encryption ciphertext, then $\mathcal{B}$ computes ct^* as a normal PLBE ciphertext; otherwise it computes ct^* as a PLBE ciphertext for index 0. Thus, $\mathcal{B}$ perfectly simulates the 1-query normal hiding security game for $\mathcal{A}$. As a result, if $\mathcal{A}$'s advantage is nonnegligible, then $\mathcal{B}$ breaks the 1-query restricted accept indistinguishability security with nonnegligible advantage. This completes the proof. ■

Index hiding security

Lemma 5.4.2. *If* Mixed-FE = (Mixed.Setup, Mixed.Enc, Mixed.SK-Enc, Mixed.Dec, Mixed.KeyGen) *is a mixed FE scheme satisfying the* 1*-query restricted function indistinguishability* (Definition 4.2.2) *property, then* PLBE = (Setup, Enc, Dec, Enc-index) *is a PLBE scheme satisfying the* 1*-query index hiding security property per* Definition 5.1.2.

Proof. The proof of this lemma is similar to that of Lemma 5.4.1, with the additional modification that the reduction algorithm now guesses the indices i, i^* on which the PLBE adversary makes its encryption query and challenge query, respectively, and the reduction algorithm aborts if its guess is incorrect. This leads to a polynomial loss ($\approx 1/n^2$) in the reduction algorithm's advantage.[2]

[2]Due to the fact that the reduction algorithm has to guess the index, we can only extend

Suppose there exists an adversary $\mathcal{A}$ such that $\mathcal{A}$'s advantage in the 1-query index hiding security game is nonnegligible. We construct an algorithm $\mathcal{B}$ that can distinguish between two secret key encryptions, therefore breaking the 1-query restricted function indistinguishability security of the mixed FE scheme.

The reduction algorithm $\mathcal{B}$ receives 1^n from $\mathcal{A}$. It sets $\kappa, \widetilde{\kappa}$ as in the construction. Next, it guesses the challenge index $i^* \in \{0, \ldots, n-1\}$ and query index $i \in \{0, \ldots, n\}$.[3] It sends κ as the functionality index and $(\mathsf{comp}_{i^*}, \mathsf{comp}_{i^*+1})$ (i.e., comparison with i^* and i^*+1) as its challenge functions to the mixed FE challenger. The challenger generates a key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$ and sends $\mathsf{mixed.pp}$ as the public parameters and challenge ciphertext $\mathsf{ct}^*_{\mathsf{attr}}$ to $\mathcal{B}$. Next, $\mathcal{B}$ makes an encryption query for function comp_i. Let the challenger's response be ciphertext $\mathsf{ct}_{\mathsf{attr}}$. $\mathcal{B}$ then queries the challenger on $n-1$ messages $j (\in [n] \setminus \{i^*+1\})$ for corresponding mixed FE secret keys and receives back keys $\mathsf{mixed.sk}_j$ for each j. It then chooses an ABE key pair $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$ and computes $n-1$ ABE keys as $\mathsf{abe.sk}_j \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}, C_{\mathsf{mixed.sk}_j})$. Next, it sends $(\mathsf{abe.pp}, \mathsf{mixed.pp})$ and $\{\mathsf{abe.sk}_j\}_{j \in [n] \setminus \{i^*+1\}}$ as the PLBE public parameters and secret keys to $\mathcal{A}$. After receiving all the keys, $\mathcal{A}$ sends its challenge message m^* to $\mathcal{B}$, and it can

the current analysis to prove q-query PLBE (adaptive) security assuming q-query mixed FE (restricted) security for constant q. However, we would like to point out that one could prove q-query PLBE *selective* security directly from q-query mixed FE (restricted) security without any security loss.

[3] Basically, the reduction algorithm guesses two things: First, it guesses the index hiding challenger with which $\mathcal{A}$ interacts and wins with nonnegligible probability; second, it guesses the index on which adversary $\mathcal{A}$ queries the PLBE challenger for index encryption.

also make an encryption query for message m on index $\tilde{i}$. Here $\mathcal{A}$ is allowed to make the encryption query either before or after the challenge query. The reduction algorithm $\mathcal{B}$ proceeds as follows. If $i \neq \tilde{i}$, $\mathcal{B}$ aborts and sends a random bit as its guess to the mixed FE challenger. Otherwise, it responds to each query as follows. $\mathcal{B}$ encrypts message m as $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$ and sends ct to $\mathcal{A}$ as its response to the encryption query. Also, it computes ciphertext ct^* as $\mathsf{ct}^* \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}^*_{\mathsf{attr}}, m^*)$ and sends ct^* as the challenge ciphertext to $\mathcal{A}$. Note that $\mathcal{A}$ could instead have sent its challenge query before sending the index encryption query. Also, $\mathcal{B}$ does not need to query the mixed FE challenger for answering any query at this point as it already has ciphertexts $\mathsf{ct}_{\mathsf{attr}}, \mathsf{ct}^*_{\mathsf{attr}}$. Finally, $\mathcal{A}$ sends its guess b to $\mathcal{B}$, and $\mathcal{B}$ forwards b as its own guess.

First, note that both $\mathcal{A}$ and $\mathcal{B}$ are allowed to make at most 1 index encryption and polynomially many secret key encryption queries, respectively. Also, note that $\mathcal{B}$ sends its secret key encryption query as well as its challenge query before making making key generation queries; thus $\mathcal{B}$ is an admissible adversary in the 1-query restricted function indistinguishability game. Next, for each query made by $\mathcal{A}$, $\mathcal{B}$ queries the mixed FE challenger exactly once, and thus all the queries are honestly and exactly answered. Finally, note that if the mixed FE challenger computed $\mathsf{ct}^*_{\mathsf{attr}}$ as a secret key FE ciphertext for function comp_{i^*}, then $\mathcal{B}$ computes ct^* as a PLBE ciphertext for index i^*; otherwise it computes ct^* as a PLBE ciphertext for index i^*+1. Thus, $\mathcal{B}$ perfectly simulates the 1-query index hiding security game for $\mathcal{A}$. Also, since $\mathcal{B}$ randomly guesses

the challenge index i^* as well as query index i, therefore with at least $1/n(n+1)$ probability $\mathcal{B}$'s guess will be correct; thus if $\mathcal{A}$'s advantage is (nonnegligible) ϵ, then $\mathcal{B}$ breaks 1-query restricted function indistinguishability security with (nonnegligible) advantage $\epsilon/n(n+1)$. This completes the proof. ■

Message hiding security

Lemma 5.4.3. *If* $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ *is a selectively secure ABE scheme per* Definition 3.5.2*, then* $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Enc\text{-}index})$ *is a PLBE scheme satisfying the* 1*-query message hiding security property per* Definition 5.1.3.

Proof. Suppose there exists an adversary $\mathcal{A}$ such that $\mathcal{A}$'s advantage in the 1-query message hiding security game is nonnegligible. We construct an algorithm $\mathcal{B}$ that can distinguish between ABE encryptions of two different messages, therefore breaking the security of the ABE scheme.

The reduction algorithm receives 1^n from $\mathcal{A}$. It sets the parameters $\kappa, \widetilde{\kappa}$ as in the construction, and starts by choosing mixed FE parameters as $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$. It then computes $\mathsf{ct}^*_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_n)$ and sends to the ABE challenger $1^{\widetilde{\kappa}}$ and $\mathsf{ct}^*_{\mathsf{attr}}$ as its challenge attribute. The ABE challenger generates a key pair $(\mathsf{abe.pp}, \mathsf{abe.sk})$ and sends $\mathsf{abe.pp}$ to $\mathcal{B}$. For $i \leq n$, $\mathcal{B}$ generates mixed FE secret keys as $\mathsf{mixed.sk}_i \leftarrow \mathsf{Mixed.KeyGen}(\mathsf{mixed.msk}, i)$ and sends $C_{\mathsf{mixed.sk}_i}$ as a predicate query to the ABE challenger and receives back secret key $\mathsf{abe.sk}_i$.

Next, it sends $(\mathsf{abe.pp}, \mathsf{mixed.pp})$ and $\{\mathsf{abe.sk}_i\}_{i \leq n}$ as the PLBE public parameters and secret keys to $\mathcal{A}$. After receiving all the keys, $\mathcal{A}$ makes a single index encryption query (m, j) to $\mathcal{B}$. $\mathcal{B}$ answers it by computing ciphertexts $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_j)$ and $\mathsf{ct} \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}, \mathsf{ct}_{\mathsf{attr}}, m)$ and sends ct to $\mathcal{A}$ as its response. $\mathcal{A}$ also sends two challenge messages (m_0^*, m_1^*) to $\mathcal{B}$. $\mathcal{B}$ then forwards (m_0^*, m_1^*) as its challenge messages to ABE challenger. Next, $\mathcal{B}$ forwards the challenge ciphertext ct^* it receives from ABE challenger to $\mathcal{A}$. Note that $\mathcal{A}$ could instead have sent its challenge query before sending the index encryption query. In that case, the reduction algorithm simply answers that first. Finally, $\mathcal{A}$ sends its guess b to $\mathcal{B}$, and $\mathcal{B}$ forwards b as its own guess.

First, note that the challenge attribute $\mathsf{ct}^*_{\mathsf{attr}}$ on each predicate $(C_{\mathsf{mixed.sk}_i})$ queried made by $\mathcal{B}$ evaluates to 0, with all but negligible probability. This follows from the correctness condition of the mixed FE system as $\mathsf{ct}^*_{\mathsf{attr}}$ encrypts function comp_n and for all $i \leq n$, $\mathsf{comp}_n(i) = 0$, and thus decrypting $\mathsf{ct}^*_{\mathsf{attr}}$ using $\mathsf{mixed.sk}_i$ outputs 0. With all-but-negligible probability, reduction algorithm $\mathcal{B}$ is therefore an admissible adversary in the ABE security game. Thus, $\mathcal{B}$ perfectly simulates the 1-query[4] message hiding security game for $\mathcal{A}$. As a result, if $\mathcal{A}$'s advantage is nonnegligible, then $\mathcal{B}$ breaks ABE security with nonnegligible advantage. This completes the proof. ■

[4]We would like to point out that the current construction actually gives a PLBE scheme that satisfies the q-query message hiding security property for arbitrary q, i.e., the number of queries need not be bounded, as long as the ABE scheme is not q-query selectively secure.

Chapter 6

A new LWE toolkit

In Chapter 3, we defined the notion of lattice trapdoors along with certain well-sampledness properties. Recall that using lattice trapdoors, it is easy to compute a preimage $\mathbf{U}$ of any matrix $\mathbf{Z}$ with respect to a matrix $\mathbf{A}$ given the trapdoor information generated while sampling $\mathbf{A}$. For any matrix $\mathbf{U}$ sampled as above, we say $\mathbf{U}$ targets $\mathbf{A}$ to matrix $\mathbf{Z}$. Now the well-sampledness properties (at a high level) state that a matrix sampled using the $\mathsf{TrapGen}$ algorithm looks uniformly random when not given the trapdoor information, and a preimage matrix that targets to a random matrix looks like a matrix with entries drawn from a Gaussian distribution.

In this chapter, we introduce certain enhanced security properties for lattice trapdoors that will be useful later in proving security of our Mixed FE system. We also provide a generic construction of lattice trapdoors that achieves these enhanced properties from any lattice trapdoor scheme that satisfies the well-sampledness properties described above. At a very high level, the enhanced security properties state the following: (1) For any matrix $\mathbf{A}$ that is sampled using the $\mathsf{TrapGen}$ algorithm, all those rows of $\mathbf{A}$ which are only used to target random rows look like random rows themselves (when not

given the trapdoor information). (2) Two preimage matrices $\mathbf{U}_0, \mathbf{U}_1$ that target a matrix $\mathbf{A}$ to different matrices $\mathbf{Z}_0, \mathbf{Z}_1$ should look indistinguishable to any adversary even when the adversary is given those rows of $\mathbf{A}$ where $\mathbf{Z}_0, \mathbf{Z}_1$ are identical. We point out that due to technical constraints in the proof of property (2) we chose to define the enhanced properties w.r.t. matrices instead of vectors.

6.1 Enhanced lattice trapdoors

Let $(\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ be a pair of randomized algorithms with the following syntax:

$\mathsf{EnTrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$. The trapdoor generation algorithm takes as input n, m, q and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $T_{\mathbf{A}}$.

$\mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{z}) \to \mathbf{u}$. The preimage sampling algorithm takes as input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with its trapdoor $T_{\mathbf{A}}$, a target vector $\mathbf{z} \in \mathbb{Z}_q^n$, and a parameter σ.[1] It outputs $\mathbf{u} \in \mathbb{Z}_q^m$ such that $\mathbf{A} \cdot \mathbf{u}^T = \mathbf{z}^T$ and $\|\mathbf{u}\| \leq \sqrt{m} \cdot \sigma$.

We require these algorithms to satisfy the following properties. These

[1] As before, the preimage sampling algorithm could be easily generalized to generate preimages of matrices instead of vectors by independently running the $\mathsf{EnSamplePre}$ algorithm on each column of the matrix. Throughout this work, we overload the notation by directly giving matrices $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$ (for any k) as inputs to the $\mathsf{SamplePre}$ algorithm.

properties are captured via security games between a challenger and a computationally bounded adversary.

Notation First, we introduce some more notation. We start by defining a matrix rearrangement procedure $\mathsf{Arrange}$ which takes as input dimensions n_1, n_2, m, and two matrices $\mathbf{A} \in \mathbb{Z}_q^{n_1 \times m}, \mathbf{B} \in \mathbb{Z}_q^{n_2 \times m}$, and an *ordered* set $S \subseteq [n_1 + n_2]$ of size n_1, and it outputs a larger combined matrix $\mathbf{C} \in \mathbb{Z}_q^{(n_1+n_2) \times m}$. Concretely, $\mathbf{C} = \mathsf{Arrange}(1^{n_1}, 1^{n_2}, 1^m, \mathbf{A}, \mathbf{B}, S)$. The rearrangement procedure is defined as follows.

Let $S = \{i_1, i_2, \ldots, i_{n_1}\}$, where $i_j < i_k$ for every $1 \leq j < k \leq n_1$. Similarly, let $\overline{S} = ([n_1 + n_2] \setminus S) = \left\{i'_1, i'_2, \ldots, i'_{n_2}\right\}$ denote the (ordered) complement of set S. Now matrix $\mathbf{C}$ is obtained by appending rows of matrices $\mathbf{A}$ and $\mathbf{B}$ as follows: for $j \in [n_1]$, $\mathbf{C}[i_j] = \mathbf{A}[j]$, and for $j \in [n_2]$, $\mathbf{C}[i'_j] = \mathbf{B}[j]$. For simplicity of notation, we drop the dimensions n_1, n_2, m as explicit inputs to $\mathsf{Arrange}$ procedure throughout this section whenever clear from the context.

Additionally, we define a matrix restriction procedure $\mathsf{Restrict}$ which takes as input dimensions n, m, and matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and an *ordered* set $S \subseteq [n]$ of size ℓ, and it outputs a smaller matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell \times m}$. Concretely, $\mathbf{C} = \mathsf{Restrict}(1^n, 1^m, \mathbf{A}, S)$. The restriction procedure is defined as follows.

Let $S = \{i_1, i_2, \ldots, i_n\}$, where $i_j < i_k$ for every $1 \leq j < k \leq \ell$. Now matrix $\mathbf{C}$ is obtained by removing rows of matrix $\mathbf{A}$ which do not lie in set S. Formally, for $j \in [\ell]$, $\mathbf{C}[j] = \mathbf{A}[i_j]$. As before, for simplicity of notation, we drop the dimensions n, m as inputs whenever clear from the context.

6.1.1 Row removal property

The first property we introduce is called the *row removal* property. It is defined via an interactive security game between the challenger and an adversary. In the game, the adversary specifies matrix dimensions n, m, a set of k ($\leq n$) indices, which represent the "target" set, and the adversary must distinguish between the following scenarios.

In the first scenario, the challenger chooses an $n \times m$ matrix $\mathbf{A}$ with a trapdoor, and sends $\mathbf{A}$ to the adversary. The adversary then participates in a query phase. For each query, the adversary sends a set of k target vectors. The challenger responds by outputting a matrix $\mathbf{U}$ such that for each index i in the target set, $\mathbf{U}$ maps the ith row of $\mathbf{A}$ to one of the target vectors. The matrix $\mathbf{U}$ maps the remaining rows of $\mathcal{A}$ to uniformly random vectors.

In the second scenario, the challenger chooses a $k \times m$ matrix $\mathbf{A}$ with a trapdoor, extends $\mathbf{A}$ to dimension $n \times m$ by attaching uniformly random rows, and sends this extended matrix to the adversary. Next, the adversary sends queries, each query consisting of k target vectors. The challenger outputs a matrix $\mathbf{U}$ such that $\mathbf{U}$ maps the ith row of $\mathbf{A}$ to the ith target vector.[2]

Definition 6.1.1 (row removal property)**.** Fix any function $q : \mathbb{N} \to \mathbb{N}$ and parameter $\sigma : \mathbb{N} \to \mathbb{R}^+$. A pair of trapdoor generation algorithms $\mathsf{LT} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ is said to satisfy the (q, σ)-*row removal* property if

[2] Although one might observe some weak resemblance between our row removal property and lattice trapdoor properties used in [5, 21], we would like to point out that after a closer inspection we observe that our row removal property is different.

for any PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$

$$\mathsf{pr}^{\mathrm{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda) = \Pr\left[1 \leftarrow \mathsf{Expt}^{\mathrm{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda)\right] \leq 1/2 + \mathrm{negl}(\lambda),$$

where $\mathsf{Expt}^{\mathrm{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$ is as defined in Fig. 6.1.

$\mathsf{Expt}^{\mathrm{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda)$

1. **Setup Phase.** The adversary $\mathcal{A}$, after receiving as input the security parameter λ, sends dimensions 1^n, 1^m and a set $S \subseteq [n]$ of size k, such that $m > 2n \log q + 2\lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$, to the challenger.[a] The challenger chooses a random bit $b \leftarrow \{0,1\}$ and proceeds as follows:
   (a) If $b = 0$, then it samples two matrices $\mathbf{B}, \mathbf{P}$ as $(\mathbf{B}, T_{\mathbf{B}}) \leftarrow \mathsf{EnTrapGen}(1^k, 1^m, q)$, $\mathbf{P} \leftarrow \mathbb{Z}_q^{(n-k)\times m}$. Next, it sets $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{P}, S)$ and sends $\mathbf{A}$ to the adversary.
   (b) Otherwise, if $b = 1$, then it chooses matrix $\mathbf{A}$ as $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{EnTrapGen}(1^n, 1^m, q)$ and sends $\mathbf{A}$ to the adversary.
2. **Query Phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{k\times t}$. The challenger responds to each query as follows.
   (a) If $b = 0$, it samples matrix $\mathbf{U}$ as $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{B}, T_{\mathbf{B}}, \sigma, \mathbf{C})$ and sends $\mathbf{U}$ to the adversary.
   (b) Otherwise if $b = 1$, it chooses a random matrix $\mathbf{Q}$ as $\mathbf{Q} \leftarrow \mathbb{Z}_q^{(n-k)\times t}$ and sets matrix $\mathbf{D}$ as $\mathbf{D} = \mathsf{Arrange}(\mathbf{C}, \mathbf{Q}, S)$. Next, it samples matrix $\mathbf{U}$ as $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{D})$ and sends $\mathbf{U}$ to the adversary.
3. $\mathcal{A}$ sends its guess b'. The experiment outputs 1 iff $b = b'$.

[a] The parameter constraints could be taken out of the security games, but for technical reasons we include them as part of the definition itself.

Figure 6.1: Experiment $\mathsf{Expt}^{\mathrm{row-rem},q,\sigma}_{\mathsf{LT},\mathcal{A}}$

A weaker row removal property is one where, in each query, the adversary is restricted to choosing a set S of size $n-1$ during setup phase, and now during query phase it must make preimage queries of the form $(1^t, \mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t})$. A different way to represent the set S in this case is by a single index $i \in [n]$ such that $\{i\} = [n] \setminus S$. We call this property the single row removal property. In our construction, we will first show a scheme that satisfies the single row removal property and then, via a simple hybrid argument, we show that the single row removal property implies the row removal property.

6.1.2 Target switching property

Next, we introduce the *target switching* property. For any target $\mathbf{Z}$, if we choose a matrix $\mathbf{B}$ with a trapdoor and output *only* the preimage of $\mathbf{Z}$ with respect to $\mathbf{B}$, then this preimage looks like a random low-norm matrix. The target switching property is an extension of this property and is captured via a security game between a challenger and an adversary. In this game, the challenger specifies the matrix dimensions n, m and a set $S \subseteq [n]$ of size k. The challenger chooses an $n \times m$ matrix $\mathbf{B}$ and sends the rows of $\mathbf{B}$ corresponding to the set S. It also chooses a challenge bit b which is used in the query phase.

Next, the adversary is allowed polynomially many queries. In each query, the adversary specifies two matrices, $\mathbf{Z}_0, \mathbf{Z}_1$, such that for every index $i \in S$, the ith rows of $\mathbf{Z}_0$ and $\mathbf{Z}_1$ are identical. The challenger outputs a matrix $\mathbf{U}$ such that for every $i \in S$, $\mathbf{U}$ maps the ith row of $\mathbf{B}$ to the ith row of $\mathbf{Z}_0$

(which is equal to the ith row of $\mathbf{Z}_1$). For the remaining indices $i \notin S$, $\mathbf{U}$ *approximately* maps the ith row of $\mathbf{A}$ to the ith row of $\mathbf{Z}_b$. Intuitively, since the adversary does not have the rows indexed by $\overline{S}$, the challenger can switch the targets from rows of $\mathbf{Z}_0$ to $\mathbf{Z}_1$.

Definition 6.1.2 (target switching property)**.** Fix any function $q : \mathbb{N} \to \mathbb{N}$, noise distribution family $\{\chi(\lambda)\}_{\lambda\in\mathbb{N}}$, and parameter $\sigma : \mathbb{N} \to \mathbb{R}^+$. A pair of trapdoor generation algorithms $\mathsf{LT} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ is said to satisfy the (q, χ, σ)-*target switching* property if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q = q(\lambda)$, $\chi = \chi(\lambda)$, $\sigma = \sigma(\lambda)$,

$$\mathsf{pr}^{\mathrm{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda) = \Pr\left[1 \leftarrow \mathsf{Expt}^{\mathrm{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\lambda)\right] \leq 1/2 + \mathrm{negl}(\lambda),$$

where $\mathsf{Expt}^{\mathrm{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$ is as defined in Fig. 6.2.

As before, we will introduce a weaker notion called the single target switching property, where in each query the adversary is restricted to outputting only a single index $i \in [n]$, and $\mathbf{Z}_0$ and $\mathbf{Z}_1$ must agree on all indices $j \neq i$. We will first show that our construction satisfies the single target switching property, and then, via a hybrid argument, we show that single target switching implies the target switching property.

6.2 Our construction of enhanced lattice trapdoors

Let $q : \mathbb{N} \to \mathbb{N}, \sigma : \mathbb{N} \to \mathbb{R}^+$ be functions, and let $\mathsf{LT} = (\mathsf{TrapGen}, \mathsf{SamplePre})$ be a pair of algorithms that satisfy q-well-sampledness of matrix (Defini-

$\mathsf{Expt}^{\mathrm{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$

1. **Setup Phase.** The adversary $\mathcal{A}$, after receiving as input the security parameter λ, sends dimensions 1^n, 1^m, set $S \subseteq [n]$ of size k, such that $m > 2n \log q + 12\lambda \cdot \log q$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$, to the challenger.[a] The challenger chooses a random bit $b \in \{0,1\}$ and proceeds as follows:
   (a) It samples matrix $\mathbf{A}$ as $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{EnTrapGen}(1^n, 1^m, q)$.
   (b) Next, it sets $\mathbf{B} = \mathsf{Restrict}(\mathbf{A}, S)$ and sends $\mathbf{B}$ to the adversary.
2. **Query Phase.** The adversary makes polynomially many queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ and $\mathsf{Restrict}(\mathbf{Z}_0, S) = \mathsf{Restrict}(\mathbf{Z}_1, S)$. The challenger responds to each query as follows:
   (a) It chooses matrix $\mathbf{E}$ as $\mathbf{E} \leftarrow \chi^{(n-k)\times t}$ and sets $\mathbf{F} = \mathsf{Arrange}(\mathbf{0}^{k\times t}, \mathbf{E}, S)$.
   (b) Next, it samples matrix $\mathbf{U}$ as $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \sigma, \mathbf{Z}_b + \mathbf{E})$ and sends $\mathbf{U}$ to the adversary.
3. $\mathcal{A}$ sends its guess b', and the experiment outputs 1 iff $b = b'$.

[a] The parameter constraints could be taken out of the security games, but for technical reasons we include them as part of the definition itself.

Figure 6.2: $\mathsf{Expt}^{\mathrm{switch},q,\chi,\sigma}_{\mathsf{LT},\mathcal{A}}(\cdot)$

tion 3.2.2), (q, σ)-preimage sampling (Definition 3.2.3). We will construct enhanced lattice trapdoors $\mathsf{LT}_{\mathsf{en}} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ using LT as follows. The construction is reminiscent of the trapdoor extension algorithms of [3, 40].

$\mathsf{EnTrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$. The trapdoor generation algorithm samples two matrices, $\mathbf{A}_1, \mathbf{A}_2$, of dimensions $n \times \lceil m/2 \rceil$ and $n \times \lfloor m/2 \rfloor$ as follows:

$$(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q),$$

$$(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q).$$

It appends both these matrices columnwise to obtain a larger matrix as $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$, and it sets the trapdoor $T_{\mathbf{A}}$ to be the combined trapdoor information $T_{\mathbf{A}} = (T_{\mathbf{A}_1}, T_{\mathbf{A}_2})$.

$\mathsf{EnSamplePre}(\mathbf{A}, T_{\mathbf{A}}, \mathbf{Z}, \sigma) \rightarrow \mathbf{U}$. The preimage sampling algorithm takes as input $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$, trapdoor $T_{\mathbf{A}} = (T_{\mathbf{A}_1}, T_{\mathbf{A}_2})$, parameter σ, and matrix $\mathbf{Z} \in \mathbb{Z}_q^{n \times k}$. It chooses a uniformly random matrix $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times k}$ and sets $\mathbf{Y} = \mathbf{Z} - \mathbf{W}$. Next, it computes matrices $\mathbf{U}_1 \in \mathbb{Z}_q^{\lceil m/2 \rceil \times k}, \mathbf{U}_2 \in \mathbb{Z}_q^{\lfloor m/2 \rfloor \times k}$ as

$$\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}),$$
$$\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}).$$

Finally, it computes the final output matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times k}$ by rowwise appending matrices $\mathbf{U}_1$ and $\mathbf{U}_2$. Concretely, $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$.

Correctness Correctness follows directly from the correctness of LT.

6.3 Proving security of $\mathsf{LT}_{\mathsf{en}}$

We will now prove that our enhanced trapdoor sampling scheme satisfies the preimage sampling, row removal, and target switching properties. First, we show that it satisfies the preimage sampling property if the underlying trapdoor scheme satisfies the preimage sampling property.

Theorem 6.3.1. *Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{R}^+$. If LT satisfies (q, σ)-preimage sampling* (Definition 3.2.3)*, then $\mathsf{LT}_{\mathsf{en}}$ also satisfies (q, σ)-preimage sampling.*

Proof Sketch. This follows directly from our construction. The preimage sampling requires that the preimage of a uniformly random matrix $\mathbf{Z}$ looks like a Gaussian sample with parameter σ. In our construction, the preimage of a random matrix $\mathbf{Z}$ consists of preimages of $\mathbf{W}$ and $\mathbf{Z} - \mathbf{W}$, where $\mathbf{W}$ is uniformly random. Since $\mathbf{Z}$ is random, so is $\mathbf{Z} - \mathbf{W}$. As a result, using the preimage sampling property of LT, we can argue that these two preimages look like two matrices drawn from $\mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times k}$ and $\mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times k}$, respectively.

6.3.1 Row removal property

Now we prove that our trapdoor sampling scheme satisfies the row removal property. Formally, we prove the following.

Theorem 6.3.2. *Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{R}^+$. If LT satisfies (q, σ)-preimage sampling* (Definition 3.2.3) *and q-well-sampledness of the matrix* (Definition 3.2.2)*, then $\mathsf{LT}_{\mathsf{en}}$ also satisfies the (q, σ)-single row removal property* (Definition 6.1.1)*.*

Proof. Our proof follows from a sequence of hybrid experiments. We start by defining a sequence of hybrid experiments such that the first and last experiments correspond to the original row removal security game when the challenger chooses its challenge bit b to be 0 and 1, respectively. To complete

the proof we show that the adversary's advantage must be negligible between any two consecutive hybrids.

We now define hybrids H_x for $x \in \{0, 1, \ldots, 12\}$. In all the hybrid experiments below, we set $q = q(\lambda)$ and $\sigma = \sigma(\lambda)$.

Hybrid H_0 This corresponds to the original game (per Definition 6.1.1, with the single row removal restriction) with $b = 0$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses matrices $(\mathbf{B}_1, T_{\mathbf{B}_1}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lceil m/2 \rceil}, q)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$ and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows:

   (a) It chooses $\mathbf{W} \leftarrow \mathbb{Z}_q^{(n-1) \times t}$ and samples $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{B}_1, T_{\mathbf{B}_1}, \sigma, \mathbf{W})$.

   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ (which is equal to $\mathbf{C} - \mathbf{W}$) and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

(c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid $\boldsymbol{H_1}$ In this experiment, the challenger chooses $\mathbf{U}_1$ to be a random Gaussian matrix with parameter σ for each query.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses matrices $(\mathbf{B}_1, T_{\mathbf{B}_1}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lceil m/2 \rceil}, q)$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$ and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows:

   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_2 In this hybrid, the challenger chooses $\mathbf{B}_1$ uniformly at random, instead of choosing it using $\mathsf{TrapGen}$. At this point, note that the left half of $\mathbf{A}$ is a uniformly random matrix.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses $\mathbf{B}_1 \leftarrow \mathbb{Z}_q^{(n-1)\times\lceil m/2\rceil}$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2\rfloor}, q)$. It sets $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It also chooses a vector $\mathbf{p} \leftarrow \mathbb{Z}_q^m$ and sets matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ as $\mathbf{A} = \mathsf{Arrange}(\mathbf{B}, \mathbf{p}, [n] \setminus \{i\})$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2\rceil\times t}$.

   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_3 This hybrid involves syntactic changes. The challenger chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and derives $\mathbf{B}_1$ by removing the ith row of $\mathbf{A}_1$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$, $(\mathbf{B}_2, T_{\mathbf{B}_2}) \leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$ and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$ and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1) \times t}$. The challenger responds to each query as follows:

   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_4 In this hybrid, the challenger chooses the left half of $\mathbf{A}$ using $\mathsf{TrapGen}$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2})$ $\leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$ and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$ and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2,$ $[n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

   (a) It samples $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

   (b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2,$ $T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid $\boldsymbol{H_5}$ In this hybrid, the challenger chooses $\mathbf{U}_1$ using $\mathsf{SamplePre}$ for each query.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

(a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2})$ $\leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$ and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

(b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$ and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

(a) It chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n\times t}$, sets $\mathbf{W} = \mathsf{Restrict}(\mathbf{W}', [n]\setminus\{i\})$, and samples $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

(b) Next, it sets $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

(c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid $\boldsymbol{H_6}$ This hybrid represents a syntactic change, in which the challenger, for each query, chooses $\mathbf{Y}$ as a uniformly random matrix, and sets $\mathbf{W} = \mathbf{C} - \mathbf{Y} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

(a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2})$ $\leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$ and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

(b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$ and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

(a) It chooses $\mathbf{Y} \leftarrow \mathbb{Z}_q^{(n-1)\times t}$ and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$.

(b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$ (which is equal to $\mathbf{C} - \mathbf{Y}$), chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

(c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_7 In this hybrid experiment, the challenger chooses $\mathbf{U}_2$ from a Gaussian distribution with parameter σ.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

(a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{B}_2, T_{\mathbf{B}_2})$ $\leftarrow \mathsf{TrapGen}(1^{n-1}, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$ and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

(b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$ and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

(a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

(b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

(c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_8 In this hybrid, the challenger chooses matrix $\mathbf{B}_2$ uniformly at random. Note that this means $\mathbf{A}_2$ is uniformly random in this hybrid.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

(a) It first chooses $\mathbf{A}_1 \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $\mathbf{B}_2 \leftarrow \mathbb{Z}_q^{(n-1)\times\lfloor m/2 \rfloor}$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$ and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

(b) It also chooses a vector $\mathbf{p}_2 \leftarrow \mathbb{Z}_q^{\lfloor m/2 \rfloor}$ and sets $\mathbf{A}_2 = \mathsf{Arrange}(\mathbf{B}_2, \mathbf{p}_2, [n] \setminus \{i\})$, $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

(a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

(b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

(c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1 \\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_9 In this hybrid, the matrix $\mathbf{A}_2$ is chosen using $\mathsf{TrapGen}$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

(a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

(b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

(a) It samples $\mathbf{U}_2 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lfloor m/2 \rfloor \times t}$.

(b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

(c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_{10} In this hybrid, the challenger chooses $\mathbf{U}_2$ using $\mathsf{SamplePre}$ for each query.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

(a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n,\ 1^{\lfloor m/2 \rfloor},\ q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2,\ [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

(b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

   (a) It chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n\times t}$ and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.

   (b) Next, it sets $\mathbf{W} = \mathbf{C} - \mathbf{B}_2 \cdot \mathbf{U}_2$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{W}' = \mathsf{Arrange}(\mathbf{W}, \mathbf{w}, [n] \setminus \{i\})$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid H_{11} This hybrid represents a syntactic change in which the ith row of matrix $\mathbf{W}'$ is set as a difference of random vector $\mathbf{c}$ and the ith row of $\mathbf{A}_2 \cdot \mathbf{U}_2$ instead of being sampled uniformly at random directly.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2 \rceil}, q\right)$, $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n,\ 1^{\lfloor m/2 \rfloor},\ q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2,\ [n] \setminus \{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

(c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

   (a) It chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n\times t}$ and samples $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.

   (b) Next, it chooses a random vector $\mathbf{c} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{C}' = \mathsf{Arrange}(\mathbf{C}, \mathbf{c}, [n]\backslash\{i\})$, sets $\mathbf{W}' = \mathbf{C}' - \mathbf{A}_2 \cdot \mathbf{U}_2$, and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W}')$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Hybrid $\boldsymbol{H_{12}}$ This hybrid represents a syntactic change. It corresponds to the security game in Definition 6.1.1 with $b = 1$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It first chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}\left(1^n, 1^{\lceil m/2\rceil}, q\right)$, $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2\rfloor}, q)$. It sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n]\setminus\{i\})$, and $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$.

   (b) It sets $\mathbf{A} = [\mathbf{A}_1 \,|\, \mathbf{A}_2]$.

   (c) Finally, it sends $\mathbf{A}$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{C})$, where $\mathbf{C} \in \mathbb{Z}_q^{(n-1)\times t}$. The challenger responds to each query as follows:

   (a) It chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n\times t}$ and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W})$.

   (b) Next, it chooses a random vector $\mathbf{c} \leftarrow \mathbb{Z}_q^t$, sets $\mathbf{C}' = \mathsf{Arrange}(\mathbf{C}, \mathbf{c}, [n] \setminus \{i\})$, sets $\mathbf{Y}' = \mathbf{C}' - \mathbf{A}_1 \cdot \mathbf{U}_1$ (which is equal to $\mathbf{C}' - \mathbf{W}'$), and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y}')$.

   (c) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. The adversary outputs a bit b'.

Analysis We will now show that any PPT adversary has at most negligible advantage in distinguishing any two consecutive hybrids. For any adversary $\mathcal{A}$, let $p_{\mathcal{A},i} : \mathbb{N} \to [0,1]$ denote the function such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i}(\lambda)$ is the probability that $\mathcal{A}$, on input 1^λ, outputs 1 in hybrid experiment H_i. From the definition of the hybrid experiments, it follows that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},12}(\lambda) = 2\mathsf{pr}^{\mathrm{row-rem},q,\sigma}_{\mathsf{LT}_{\mathsf{en}},\mathcal{A}}(\lambda) - 1$. Therefore, to show that $\mathsf{LT}_{\mathsf{en}}$ satisfies the (q,σ)-row removal property, it suffices to show that for all $\mathcal{A}$ and $i \in [12]$, there exist negligible functions negl_i such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i-1}(\lambda) - p_{\mathcal{A},i}(\lambda) \leq \mathrm{negl}_i(\lambda)$.

Lemma 6.3.1. *Assuming* LT *satisfies* (q,σ)*-preimage sampling, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl_1(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},1}(\lambda) \leq negl_1(\lambda)$.

Proof. Suppose there exist an adversary $\mathcal{A}$ and a nonnegligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},1}(\lambda) \geq \eta(\lambda)$. Moreover, let $s_{\mathcal{A}}$ denote the number of queries made by $\mathcal{A}$, and let $t_{\mathcal{A}}$ denote a bound on the number of columns in queried matrix $\mathbf{C}$ (note that the reduction algorithm is allowed to depend on the adversary, and therefore it knows $s_{\mathcal{A}}$ and $t_{\mathcal{A}}$ corresponding to $\mathcal{A}$).[3] Then we can construct a reduction algorithm $\mathcal{B}$ such that $\mathsf{pr}_{\mathsf{LT},\mathcal{B}}^{\mathrm{preimg},q,\sigma}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$.

The reduction algorithm receives n, m, index $i \in [n]$ from $\mathcal{A}$ such that $m > 2n \log q + 2\lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$. It forwards $1^{n-1}, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$ to the challenger.[4] It receives $\mathbf{B}_1 \in \mathbb{Z}_q^{(n-1)\times\lceil m/2 \rceil}$ and $\widetilde{\mathbf{U}} \in \mathbb{Z}_q^{\lceil m/2 \rceil \times (s_{\mathcal{A}} \cdot t_{\mathcal{A}})}$. Note that the trapdoor for $\mathbf{B}_1$ is not used in hybrid H_1. The reduction algorithm chooses $(\mathbf{B}_2, T_{\mathbf{B}_2})$ using $\mathsf{TrapGen}$, computes $\mathbf{A}$ as in H_0 (and H_1), and sends $\mathbf{A}$ to $\mathcal{A}$. The challenger also partitions $\widetilde{\mathbf{U}} = \left[\widetilde{\mathbf{U}}_1 \,|\, \ldots \,|\, \widetilde{\mathbf{U}}_{s_{\mathcal{A}}}\right]$, where each $\widetilde{\mathbf{U}}_j \in \mathbb{Z}_q^{\lceil m/2 \rceil \times t_{\mathcal{A}}}$.

Next, the adversary sends queries. For the i^*th query, the adversary sends $(1^t, \mathbf{C})$ for some $t \leq t_{\mathcal{A}}$. The reduction algorithm sets $\mathbf{U}_1$ to be the first t columns of $\widetilde{\mathbf{U}}_{i^*}$. It computes $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$. It sets $\mathbf{U}$ as in H_0/H_1 and sends $\mathbf{U}$ to $\mathcal{A}$.

[3] Throughout this section, we construct the nonuniform reduction algorithm, as our reduction algorithms depends on the number of queries made by the adversary as well as the size of the matrices in each query. However, we would like to point out that the reduction could be made uniform by simply guessing both of these bounds. This would result in a polynomial loss in the reduction algorithm's advantage.

[4] Note that the reduction algorithm chooses admissible parameters, since $\lceil m/2 \rceil > n \log q + \lambda > (n-1) \log q + \lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda > \sqrt{(n-1) \cdot \log q \cdot \log m/2} + \lambda$.

Finally, after all queries, if $\mathcal{A}$ outputs 1, $\mathcal{B}$ guesses that $\widetilde{\mathbf{U}}$ is sampled using $\mathsf{SamplePre}$; otherwise it guesses that $\widetilde{\mathbf{U}}$ is a random Gaussian matrix sampled with parameter σ.

Note that depending on whether $\widetilde{\mathbf{U}}$ is sampled using $\mathsf{SamplePre}$ or sampled from Gaussian distribution, $\mathcal{B}$ simulates either H_0 or H_1 perfectly. As a result, $\mathcal{B}$'s advantage in the preimage sampling game is at least $\eta(\lambda)$. ■

Lemma 6.3.2. *Assuming* LT *satisfies q-well-sampledness of matrix, for any adversary* $\mathcal{A}$*, there exists a negligible function* $negl_2(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $p_{\mathcal{A},1}(\lambda) - p_{\mathcal{A},2}(\lambda) \leq negl_2(\lambda)$*.*

Proof. Suppose there exist an adversary $\mathcal{A}$ and a nonnegligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1}(\lambda) - p_{\mathcal{A},2}(\lambda) \geq \eta(\lambda)$. Then we can construct a reduction algorithm $\mathcal{B}$ such that $\mathsf{pr}_{\mathsf{LT},\mathcal{B}}^{\mathrm{matrix},q,\sigma}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$.

The reduction algorithm receives $1^n, 1^m$, index $i \in [n]$ from $\mathcal{A}$. It forwards $1^{n-1}, 1^{\lceil m/2 \rceil}$ to the challenger.[5] It receives $\mathbf{B}_1$. The reduction algorithm chooses $(\mathbf{B}_2, T_{\mathbf{B}_2})$ using $\mathsf{TrapGen}$, sets $\mathbf{A}$ as in H_1 (and H_2), and sends $\mathbf{A}$ to $\mathcal{A}$.

Next, the adversary sends queries. For each query $\mathbf{C}$, the challenger chooses $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$ and computes $\mathbf{Y} = \mathbf{C} - \mathbf{B}_1 \cdot \mathbf{U}_1$ and $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{B}_2, T_{\mathbf{B}_2}, \sigma, \mathbf{Y})$. It chooses $\mathbf{p}$, sets $\mathbf{U}$ as in H_1/H_2, and sends $\mathbf{U}$ to $\mathcal{A}$.

[5]Note that the reduction algorithm chooses admissible parameters, since $\lceil m/2 \rceil > n \log q + \lambda > (n-1) \log q + \lambda$.

After all the queries, if $\mathcal{A}$ outputs 1, $\mathcal{B}$ guesses that $\mathbf{B}_1$ is sampled using $\mathsf{TrapGen}$; otherwise it guesses that $\mathbf{B}_1$ is a uniformly random matrix. Note that, depending on whether $\mathbf{B}_1$ is sampled using $\mathsf{TrapGen}$ or sampled uniformly at random, $\mathcal{B}$ simulates either H_1 or H_2 perfectly. As a result, $\mathcal{B}$'s advantage in the matrix well-sampledness game is at least $\eta(\lambda)$. ■

Lemma 6.3.3. *For any adversary $\mathcal{A}$, $p_{\mathcal{A},2}(\lambda) = p_{\mathcal{A},3}(\lambda)$.*

Since the only changes from H_2 to H_3 are syntactical, it follows that any adversary has identical behavior in both hybrids.

Lemma 6.3.4. *Assuming LT satisfies q-well-sampledness of matrix, for any PPT adversary $\mathcal{A}$, there exists a negligible function $negl_4(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},3}(\lambda) - p_{\mathcal{A},4}(\lambda) \leq negl_4(\lambda)$.*

This proof is identical to the proof of Lemma 6.3.2, except that the reduction algorithm must send $1^n, 1^{\lceil m/2 \rceil}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$.

Lemma 6.3.5. *Assuming LT satisfies (q, σ)-preimage sampling, for any PPT adversary $\mathcal{A}$, there exists a negligible function $negl_5(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},4}(\lambda) - p_{\mathcal{A},5}(\lambda) \leq negl_5(\lambda)$.*

This proof is identical to the proof of Lemma 6.3.1, except that the reduction algorithm must send $1^n, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$.

Lemma 6.3.6. *For any adversary $\mathcal{A}$, $p_{\mathcal{A},5}(\lambda) = p_{\mathcal{A},6}(\lambda)$.*

Note that the distributions in H_5 and H_6 are identical. In hybrid H_5, the challenger chooses $\mathbf{W}' \leftarrow \mathbb{Z}_q^{n \times m}$, derives $\mathbf{W}$ from $\mathbf{W}'$ by removing the ith

row, and sets $\mathbf{Y} = \mathbf{C} - \mathbf{W}$. In hybrid H_6, it chooses $\mathbf{Y} \leftarrow \mathbb{Z}_q^{(n-1)\times m}$, sets $\mathbf{W} = \mathbf{C} - \mathbf{Y}$, and sets $\mathbf{W}'$ to be a matrix extended from $\mathbf{W}$ by inserting a random vector at row i. The distribution of $(\mathbf{W}, \mathbf{W}', \mathbf{Y})$ is identical in both hybrid experiments.

Lemma 6.3.7. *Assuming* LT *satisfies* (q, σ)*-preimage sampling, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl_7(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $p_{\mathcal{A},6}(\lambda) - p_{\mathcal{A},7}(\lambda) \leq negl_7(\lambda)$*.*

This proof is identical to the proof of Lemma 6.3.1, except that the reduction algorithm must send $1^{n-1}, 1^{\lfloor m/2 \rfloor}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}, 1^{s_{\mathcal{A}} \cdot t_{\mathcal{A}}}$. It uses the challenger's response for setting $\mathbf{B}_2, \mathbf{U}_2$ and chooses the remaining components by itself.

Lemma 6.3.8. *Assuming* LT *satisfies q-well-sampledness of matrix, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl_8(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $p_{\mathcal{A},7}(\lambda) - p_{\mathcal{A},8}(\lambda) \leq negl_8(\lambda)$*.*

This proof is identical to the proof of Lemma 6.3.2, except that the reduction algorithm must send $1^{n-1}, 1^{\lfloor m/2 \rfloor}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$. It uses the challenger's response for setting $\mathbf{B}_2$ and chooses the remaining components by itself.

Lemma 6.3.9. *Assuming* LT *satisfies q-well-sampledness of matrix, for any adversary* $\mathcal{A}$*, there exists a negligible function* $negl_9(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $p_{\mathcal{A},8}(\lambda) - p_{\mathcal{A},9}(\lambda) \leq negl_9(\lambda)$*.*

This proof is identical to the proof of Lemma 6.3.2, except that the reduction algorithm must send $1^n, 1^{\lfloor m/2 \rfloor}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$. It uses the challenger's response for setting $\mathbf{A}_2$ and chooses the remaining components by itself.

Lemma 6.3.10. *Assuming* LT *satisfies* (q, σ)*-preimage sampling, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl_{10}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $p_{\mathcal{A},9}(\lambda) - p_{\mathcal{A},10}(\lambda) \leq negl_{10}(\lambda)$*.*

This proof is identical to the proof of Lemma 6.3.1, except that the reduction algorithm must send $1^n, 1^{\lfloor m/2 \rfloor}$ instead of $1^{n-1}, 1^{\lceil m/2 \rceil}$. It uses the challenger's response for setting $\mathbf{A}_2, \mathbf{U}_2$ and chooses the remaining components by itself.

Lemma 6.3.11. *For any adversary* $\mathcal{A}$*,* $p_{\mathcal{A},10}(\lambda) = p_{\mathcal{A},11}(\lambda)$*.*

In hybrid experiment H_{10}, the challenger chooses $\mathbf{Y}' \leftarrow \mathbb{Z}_q^{n \times m}$ and derives $\mathbf{Y}'$ by removing the ith row. It sets $\mathbf{W} = \mathbf{C} - \mathbf{Y}$, chooses a uniformly random vector $\mathbf{w} \leftarrow \mathbb{Z}_q^m$, and constructs $\mathbf{W}'$ from $\mathbf{W}$ and $\mathbf{w}$. In hybrid H_{11}, the challenger chooses $\mathbf{Y}'$ uniformly at random, extends $\mathbf{C}'$ from $\mathbf{C}$ by inserting a random vector at the ith row, and sets $\mathbf{W}' = \mathbf{C}' - \mathbf{Y}'$. As a result, $(\mathbf{Y}', \mathbf{W}')$ are identically distributed in both hybrids. The remaining components in the hybrids either are identical or can be derived from $\mathbf{Y}', \mathbf{W}'$.

Lemma 6.3.12. *For any adversary* $\mathcal{A}$*,* $p_{\mathcal{A},11}(\lambda) = p_{\mathcal{A},12}(\lambda)$*.*

Note that the distributions in H_{11} and H_{12} are identical. The proof is identical to that of Lemma 6.3.6.

Using the above lemmas, it follows that the advantage of an adversary in the row removal experiment is at most $\text{negl}(\lambda)$.

■

Theorem 6.3.3. *Fix any function $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$. Assuming $\mathsf{LT}_{\mathsf{en}} = (\mathsf{EnTrapGen}, \mathsf{EnSamplePre})$ satisfies the (q, σ)-single row removal property, $\mathsf{LT}_{\mathsf{en}}$ also satisfies the (q, σ)-row removal property.*

Proof. This proof follows via a simple hybrid argument, where we gradually remove the nontargeted rows from $\mathbf{A}$ one by one. Suppose $\mathcal{A}$ outputs set S of size k. We will define $n - k + 1$ hybrids $H_1, \ldots, H_{n-k+1}$ as follows.

Hybrid H_i for $0 \leq i \leq n - k$ In hybrid H_i, the challenger does the following:

1. Let $(1^n, 1^m, S \subseteq [n]) \leftarrow \mathcal{A}(1^\lambda)$, and let $S = \{i_1, \ldots, i_k\}$, $\overline{S} = \{i'_1, \ldots, i'_{n-k}\}$. Let $S_i = S \cup \{i'_1, \ldots, i'_i\} = \{\tilde{i}_1, \ldots, \tilde{i}_{k+i}\}$ and $\overline{S_i} = [n] \backslash S_i = \{\tilde{i'}_1, \ldots, \tilde{i'}_{n-k-i}\}$.
2. It chooses $(\mathbf{B}, T_{\mathbf{B}}) \leftarrow \mathsf{EnTrapGen}(1^{k+i}, 1^m, q)$, $\mathbf{P} \leftarrow \mathbb{Z}_q^{(n-k-i)\times m}$.
3. It sets $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, where $\mathbf{A}[\tilde{i}_j] = \mathbf{B}[j]$ for all $j \leq k + i$, and $\mathbf{A}[\tilde{i'}_j] = \mathbf{P}[j]$ for all $j \leq n - k - i$, and it sends $\mathbf{A}$ to $\mathcal{A}$.

4. Next, the adversary sends queries. For each query $\{\mathbf{c}_j\}_{j\in S}$, the challenger first chooses $\mathbf{c}_j \leftarrow \mathbb{Z}_q^t$ for each $j \in S_i \setminus S$, and sets $\mathbf{C} \in \mathbb{Z}_q^{(k+i)\times t}$, where $\mathbf{C}[j] = \mathbf{c}_{\tilde{i}_j}$ for all $j \leq k+i$.

5. Next, it chooses $\mathbf{U} \leftarrow \mathsf{EnSamplePre}(\mathbf{B}, T_{\mathbf{B}}, \mathbf{C}, \sigma)$ and sends $(\mathbf{A}, \mathbf{U})$.

6. Finally, after all queries, the adversary $\mathcal{A}$ outputs a bit b'.

Note that H_0 and H_{n-k} correspond to $b = 0$ and $b = 1$ in the row removal experiment. For any adversary $\mathcal{A}$, let $p_{\mathcal{A},i}(\lambda)$ denote the probability of $\mathcal{A}$ outputting 1 in hybrid H_i. We will show that for any adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that $p_{\mathcal{A},i}(\lambda)$ and $p_{\mathcal{A},i+1}$ differ by at most $\mathrm{negl}(\lambda)$.

Lemma 6.3.13. *Fix any index $i \in \{0, 1, \ldots, n-k-1\}$. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the single row removal property, for any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i}(\lambda) - p_{\mathcal{A},i+1}(\lambda) \leq negl(\lambda)$.*

Proof. Suppose there exist an adversary $\mathcal{A}$ and a nonnegligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},i}(\lambda) - p_{\mathcal{A},i+1}(\lambda) \geq \eta(\lambda)$. We can use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that can break the single row removal property with advantage $\eta(\cdot)$.

The reduction algorithm first receives $1^n, 1^m, S = \{i_1, \ldots, i_k\}$. It defines $S_{i+1} = S \cup \{i'_1, \ldots, i'_{i+1}\} = \{\tilde{i}_1, \ldots, \tilde{i}_{k+i+1}\}$, and lets $\mathsf{indx} \in [k+i+1]$ be the index such that $\tilde{i}_{\mathsf{indx}} = i'_{i+1}$. The reduction algorithm sends $1^{k+i+1}, 1^m$

and indx.[6] It receives $\mathbf{B}$ from the challenger. The reduction algorithm sets $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ such that for each $j \leq k+i+1$, $\mathbf{A}[\tilde{i}_j] = \mathbf{B}[j]$, and the remaining rows are chosen uniformly at random.

For each query $\{\mathbf{c}_j\}_{j \in S}$, the reduction algorithm chooses vectors $\{\mathbf{c}_{i'_1}, \ldots, \mathbf{c}_{i'_i}\}$ uniformly at random from $\mathbb{Z}_q^t$ and sends $\{\mathbf{c}_{\tilde{i}_j}\}_{j \in [k+i+1], j \neq \mathsf{indx}}$ to the challenger. The challenger sends $\mathbf{U}$ to the reduction algorithm. The reduction algorithm forwards $\mathbf{U}$ to $\mathcal{A}$. Finally, after all the queries, $\mathcal{B}$ outputs the adversary's final output bit.

Clearly, the reduction algorithm simulates either H_i or H_{i+1}, depending on the challenger's output, and therefore the advantage of $\mathcal{B}$ is $p_{\mathcal{A},i}(\lambda) - p_{\mathcal{A},i+1}(\lambda)$. ■

■

6.3.2 Target switching property

Now we prove that our trapdoor sampling scheme satisfies the target switching property. Formally, we prove the following.

Theorem 6.3.4. *Fix any functions $q : \mathbb{N} \to \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{R}^+$, and error distribution family $\{\chi(\lambda)\}_\lambda$. If LT satisfies q-well-distributedness of matrix* (Definition 3.2.2)*, (q, σ)-well-distributedness of preimage* (Definition 3.2.3)*, and $\mathsf{LWE\text{-}sp}_{(d,q,\sigma,\chi)}$ holds (LWE with short public vectors; Assumption 3) where*

[6] Note that the reduction algorithm chooses admissible parameters, since $m > n \log q + \lambda > (k+i+1) \log q + \lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda > \sqrt{(k+i+1) \cdot \log q \cdot \log m} + \lambda$.

$d(\lambda) = 6\lambda \log q(\lambda)$, *then* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* (q, σ, χ)*-single target switching property.*

Proof. To prove the above theorem, we first define a sequence of hybrid games where the first game is the single target switching security game, and in the last game the adversary's advantage is exactly 0. Later we show that the adversary's advantage in any two consecutive hybrid games is negligible. For simplicity of notation, we will let $d = d(\lambda)$, $q = q(\lambda)$, $\sigma = \sigma(\lambda)$, and $\chi = \chi(\lambda)$.

Hybrid H_0 This corresponds to the single target switching security game.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q)$ and $(\mathbf{A}_2, T_{\mathbf{A}_2})$ $\leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0, 1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$, and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0,$
   $[n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows:

   (a) It chooses $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times t}$ and computes $\mathbf{U}_1 \leftarrow \mathsf{SamplePre}(\mathbf{A}_1, T_{\mathbf{A}_1}, \sigma, \mathbf{W})$.

(b) It also samples vector $\mathbf{e} \leftarrow \chi^t$ and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1)\times t}, \mathbf{e}, [n] \setminus \{i\})$.

(c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$ (which is equal to $\mathbf{Z}_b - \mathbf{W} + \mathbf{E}$) and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

(d) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix}\mathbf{U}_1\\ \mathbf{U}_2\end{smallmatrix}\right]$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_1 In this hybrid experiment, the challenger sets $\mathbf{U}_1$ to be a Gaussian matrix for each query.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It chooses matrices $(\mathbf{A}_1, T_{\mathbf{A}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lceil m/2 \rceil}, q)$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0,1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$ and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n\times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0,$
   $[n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows:

   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

(b) It also samples vector $\mathbf{e} \leftarrow \chi^t$ and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1)\times t}, \mathbf{e}, [n] \setminus \{i\})$.

(c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

(d) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_2 In this hybrid experiment, the challenger sets $\mathbf{A}_1$ to be a uniformly random matrix (that is, sampled without a trapdoor).

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0, 1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$ and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0,$
   $[n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows:

   (a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

(b) It also samples vector $\mathbf{e} \leftarrow \chi^t$ and sets $\mathbf{E} = \mathsf{Arrange}(\mathbf{0}^{(n-1)\times t}, \mathbf{e}, [n] \setminus \{i\})$.

(c) Next, it sets $\mathbf{Y} = \mathbf{Z}_b - \mathbf{A}_1 \cdot \mathbf{U}_1 + \mathbf{E}$ and computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

(d) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_3 This hybrid is a syntactic change. Here, we express $\mathbf{Y}$ in terms of $\mathbf{B}_1$ and the ith row of $\mathbf{A}_1$. Note that the ith row of $\mathbf{A}_1$ is used only for computing the ith row of $\mathbf{Y}$.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0, 1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$ and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0,$
   $[n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows:

(a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

(b) It also samples vector $\mathbf{e} \leftarrow \chi^t$ and sets $\mathbf{Z}'_b = \mathsf{Restrict}(\mathbf{Z}_b, [n] \setminus \{i\})$.

(c) Next, it sets $\mathbf{Y}' = \mathbf{Z}'_b - \mathbf{B}_1 \cdot \mathbf{U}_1$, $\mathbf{y} = \mathbf{Z}_b[i] - \mathbf{A}_1[i] \cdot \mathbf{U}_1 + \mathbf{e}$, and $\mathbf{Y} = \mathsf{Arrange}(\mathbf{Y}', \mathbf{y}, [n] \setminus \{i\})$. It then computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

(d) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Hybrid H_4 In this hybrid experiment, the challenger sets the ith row of $\mathbf{Y}$ to be a uniformly random vector.

1. **Setup phase.** The adversary $\mathcal{A}$ sends $1^n, 1^m$, index $i \in [n]$. The challenger proceeds as follows:

   (a) It chooses $\mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times \lceil m/2 \rceil}$ and $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$. It also chooses a random bit $b \leftarrow \{0, 1\}$.

   (b) Next, it sets $\mathbf{B}_1 = \mathsf{Restrict}(\mathbf{A}_1, [n] \setminus \{i\})$, $\mathbf{B}_2 = \mathsf{Restrict}(\mathbf{A}_2, [n] \setminus \{i\})$ and sends $[\mathbf{B}_1 \,|\, \mathbf{B}_2]$ to $\mathcal{A}$.

2. **Query phase.** The adversary makes a polynomial number of preimage queries of the form $(1^t, \mathbf{Z}_0, \mathbf{Z}_1)$, where $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n \times t}$ such that $\mathsf{Restrict}(\mathbf{Z}_0, [n] \setminus \{i\}) = \mathsf{Restrict}(\mathbf{Z}_1, [n] \setminus \{i\})$. The challenger responds to each query as follows:

(a) It computes $\mathbf{U}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{\lceil m/2 \rceil \times t}$.

(b) It sets $\mathbf{Z}'_b = \mathsf{Restrict}(\mathbf{Z}_b, [n] \setminus \{i\})$.

(c) Next, it sets $\mathbf{Y}' = \mathbf{Z}'_b - \mathbf{B}_1 \cdot \mathbf{U}_1$, $\mathbf{y} \leftarrow \mathbb{Z}_q^t$, and $\mathbf{Y} = \mathsf{Arrange}(\mathbf{Y}', \mathbf{y}, [n] \setminus \{i\})$. It then computes $\mathbf{U}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}_2, T_{\mathbf{A}_2}, \sigma, \mathbf{Y})$.

(d) Finally, it sends $\mathbf{U} = \left[\begin{smallmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{smallmatrix}\right]$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs its guess b'.

Analysis We will now analyze the adversary's advantage in the single target switching experiment. Let $p_{\mathcal{A},l}(\lambda)$ denote the probability of $\mathcal{A}$ guessing correctly (i.e., $b' = b$) at the end of hybrid experiment H_l. We will show that for every PPT adversary $\mathcal{A}$ and $l \in [4]$ there exist negligible functions $\mathrm{negl}_l(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_{\mathcal{A},l-1} - p_{\mathcal{A},l} \leq \mathrm{negl}_i(\lambda)$.

Lemma 6.3.14. *Assuming* LT *satisfies* (q, σ)*-preimage sampling, for any adversary* $\mathcal{A}$ *there exists a negligible function* $negl_1(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0} - p_{\mathcal{A},1} \leq negl_1(\lambda)$.

The proof of this lemma is similar to the proof of Lemma 6.3.1.

Lemma 6.3.15. *Assuming* LT *satisfies* q*-well-sampledness of matrix, for any adversary* $\mathcal{A}$ *there exists a negligible function* $negl_2(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1} - p_{\mathcal{A},2} \leq negl_2(\lambda)$.

The proof of this lemma is similar to the proof of Lemma 6.3.2.

Lemma 6.3.16. *For any adversary* $\mathcal{A}$, $p_{\mathcal{A},2} = p_{\mathcal{A},3}$.

Note that the only differences in H_2 and H_3 are syntactic changes with respect to matrix $\mathbf{Y}$. As a result, the distributions in the two hybrids are identical.

Lemma 6.3.17. *If* $\mathsf{LWE\text{-}sp}_{(d,q,\sigma,\chi)}$ *holds (Assumption 3) where* $d(\lambda) = 6\lambda \log q(\lambda)$*, then for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl_4(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $p_{\mathcal{A},3} - p_{\mathcal{A},4} \leq negl_4(\lambda)$*.*

Proof. Suppose, on the contrary, there exist an adversary $\mathcal{A}$ and a nonnegligible function $\eta(\cdot)$ such that $p_{\mathcal{A},3} - p_{\mathcal{A},4} \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$. We will construct a reduction algorithm $\mathcal{B}$ such that $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE\text{-}sp},d,q,\sigma,\chi}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$.

The reduction algorithm first receives $1^n, 1^m, i \in [n], \mathbf{Z}_0, \mathbf{Z}_1$ from the adversary $\mathcal{A}$ (such that $m > 2n \log q + 12\lambda \log q$). The reduction algorithm chooses $\mathbf{B}_1 \leftarrow \mathbb{Z}_q^{(n-1)\times m}$, $(\mathbf{A}_2, T_{\mathbf{A}_2}) \leftarrow \mathsf{TrapGen}(1^n, 1^{\lfloor m/2 \rfloor}, q)$ and derives $\mathbf{B}_2$ from $\mathbf{A}_2$ by removing the ith row. It defines $\mathbf{B} = [\mathbf{B}_1 \,|\, \mathbf{B}_2]$ and sends it to the adversary. It also chooses a random bit $b \leftarrow \{0, 1\}$.

Next, the reduction algorithm receives queries from the adversary, and it uses the $\mathsf{LWE\text{-}sp}$ challenger to define matrices $\mathbf{U}_1$ and $\mathbf{y}$. For each query, the adversary sends two matrices, $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{n\times t}$, such that all their rows are equal, except the ith one. The reduction algorithm queries the $\mathsf{LWE\text{-}sp}$ challenger for t queries and receives $\{(\mathbf{a}_r, u_r)\}_{r\in[t]}$, where $\mathbf{a}_r \in \mathbb{Z}_q^d$ for each $r \in [t]$. It chooses $\tilde{\mathbf{a}}_r \leftarrow \mathcal{D}_{\mathbb{Z}_q,\sigma}^{\lceil m/2 \rceil - d}$ for each $r \in [m]$, $\tilde{\mathbf{s}} \leftarrow \mathbb{Z}_q^{\lceil m/2 \rceil - d}$.[7] Next, it sets $\mathbf{U}_1 \in \mathbb{Z}_q^{\lceil m/2 \rceil \times m}$

[7]Since $m > 2n \log q + 12\lambda \cdot \log q$ and $d = 6\lambda \log q$, thus $\lceil m/2 \rceil - d \geq 0$. Here we would like to point out that in our target switching security game the adversary is allowed to

to be a matrix whose rth column is $[\mathbf{a} \,|\, \tilde{\mathbf{a}}_r]^T$ for each $r \in [m]$. It sets $\mathbf{y} \in \mathbb{Z}_q^m$, where $\mathbf{y}_r = \mathbf{Z}_b[i]_r - u_r - \tilde{\mathbf{s}} \cdot \tilde{\mathbf{a}}_r^T$.

Once $\mathbf{y}$ and $\mathbf{U}_1$ are determined, the reduction algorithm can compute $\mathbf{U}_2$ using $\mathbf{B}_1, \mathbf{U}_1, \mathbf{Z}_b, T_{\mathbf{A}_2}$. It sets $\mathbf{B}$ and $\mathbf{U}$ as in H_3/H_4 and sends $\mathbf{U}$ to $\mathcal{A}$.

Finally, after all the queries, the adversary outputs a bit b'. If $b = b'$, the reduction algorithm guesses 0 (i.e., u_r is an LWE sample); otherwise it guesses 1 (i.e., u_r is a uniformly random element).

Now note that if the LWE-sp challenger uses oracle $O_2()$, then the reduction algorithm simulates H_4; otherwise it simulates H_3. Therefore, the advantage of $\mathcal{B}$ is at least $p_{\mathcal{A},3} - p_{\mathcal{A},4}$. ■

Using the above lemmas, it follows that any PPT adversary has an advantage at most $\text{negl}(\lambda)$ in the single target switching security game, since in the last hybrid game (H_4) the challenger's response is independent of bit b and thus any adversary advantage is exactly 0. ■

Theorem 6.3.5. *Fix any function $q : \mathbb{N} \to \mathbb{N}$, $\sigma : \mathbb{N} \to \mathbb{R}^+$ and distribution family $\{\chi(\lambda)\}_\lambda$. Assuming $\mathsf{LT}_{\mathsf{en}}$ satisfies the (q, σ, χ)-single target switching property, $\mathsf{LT}_{\mathsf{en}}$ also satisfies the (q, σ, χ)-target switching property.*

The proof of this theorem follows from a hybrid argument similar to that in the proof of Theorem 6.3.3.

---

choose the dimensions m, n; as stated in our LWE assumption framework, however, the lattice dimensions are not chosen by the adversary. Due to this definitional inconsistency, as a reduction algorithm we always choose to attack the LWE problem for dimensions $d(\lambda) = 6\lambda \log q(\lambda)$. This could be avoided by adapting the existing definitions.

Chapter 7

Constructing 1-query mixed functional encryption

In this chapter, we describe our construction of mixed FE for input-circling branching programs with polynomial width and length. Concretely, $\mathcal{M}_\kappa = \{0,1\}^k$ and $\mathcal{F}_\kappa$ denotes the class of input-circling branching programs with input space $\{0,1\}^k$, width w, and length $\ell = k \cdot L$, where $\kappa = (k, w, L)$. In other words, every branching program reads each input bit L times in a circular fashion. Before describing our construction, we introduce some shorthand notation that we will use throughout this section.

7.1 Notation

Consider a set of $4\ell\lambda$ matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}$ and $w\ell$ matrices $\{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}$, where each individual matrix lies in $\mathbb{Z}_q^{n\times m}$. For $i \in [\ell]$, let $\mathbf{D}_i$

be another matrix defined as below:

$$\mathbf{D}_i = \begin{bmatrix} \mathbf{B}_{i,0}^{(1,0)} \\ \mathbf{B}_{i,1}^{(1,0)} \\ \mathbf{B}_{i,0}^{(1,1)} \\ \vdots \\ \mathbf{B}_{i,1}^{(\lambda,1)} \\ \mathbf{P}_{i,1} \\ \vdots \\ \mathbf{P}_{i,w} \end{bmatrix}$$

The matrix $\mathbf{D}_i$ consists of matrices $\mathbf{B}_{i,b}^{(j,\beta)}$ arranged per adjoining well-defined ordering. Concretely, let (i, j, β, b) be the indices of any $\mathbf{B}$ matrix. The ordering we define is that

$$(i, j_1, \beta_1, b_1) \prec (i, j_2, \beta_2, b_2) \iff \begin{cases} j_1 < j_2, \text{ or} \\ j_1 = j_2 \wedge \beta_1 < \beta_2, \text{or} \\ j_1 = j_2 \wedge \beta_1 = \beta_2 \\ \qquad \wedge b_1 < b_2. \end{cases}$$

Thus, per our ordering $(i, 1, 0, 0)$, $(i, 1, 0, 1)$, $(i, 1, 1, 0)$, $(i, 1, 1, 1)$, $\ldots$, $(i, \lambda, 1, 1)$ is an increasing sequence of indices. Similarly, we can define an ordering for matrices $\mathbf{P}_{i,v}$ for $v \in [w]$ (i.e., $(i, v_1) \prec (i, v_2) \iff v_1 < v_2$).

In words, matrix $\mathbf{D}_i$ is defined by rowwise appending matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{j\in[\lambda],\beta,b\in\{0,1\}}$ and $\{\mathbf{P}_{i,v}\}_{v\in[w]}$ in an increasing order per the ordering "$\prec$" defined above. We will use the following shorthand notation for representing the above matrix more compactly:

$$\mathbf{D}_i = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}.$$

Similarly, for any (possibly empty) sets $S_1 \subseteq [\lambda] \times \{0,1\}^2, S_2 \subseteq [w]$, we will use the shorthand

$$\mathbf{D}_i^{S_1,S_2} = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_1} \\ \{\mathbf{P}_{i,v}\}_{v\in S_2} \end{bmatrix}$$

to represent the matrix generated by rowwise appending matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{(j,\beta,b)\in S_1}$ and $\{\mathbf{P}_{i,v}\}_{v\in S_2}$ in an increasing order per the ordering "$\prec$".

7.2 Construction

In this section, we present our Mixed FE scheme. First, we provide the parameter constraints required by our correctness and security proof. For functionality indices (k, w, L) (where k denotes the input length, and $w, \ell\ (= k \cdot L)$ are the width and length of branching programs), the setup algorithm chooses parameters n, m, q, σ and noise distributions $\chi_{\mathsf{big}}, \chi_{\mathsf{last}}, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_s, \chi_{\mathsf{lwe}}$ as follows.

Fix any $\epsilon < 1$. Let χ_1, χ_2 be a B_1-bounded discrete Gaussian distribution with parameter σ such that $B_1 = \sqrt{m} \cdot \sigma$. Also, for any $B > 0$, let U_B denote the uniform distribution on $\mathbb{Z} \cap [-B, B]$, i.e., integers between $\pm B$. The setup algorithm chooses parameters n, m, σ, q and sets noise distributions $\chi_{\mathsf{big}}, \chi_{\mathsf{last}}, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_s, \chi_{\mathsf{lwe}}$ with the following constraints. Also, we will use different versions of LWE, with different noise distributions.

- $\chi_s = \chi_{\mathsf{appr}} = \chi_{\mathsf{pre}} = \chi_1$ (for enhanced trapdoor sampling);
- $n = \mathsf{poly}(\lambda)$, $\chi_{\mathsf{lwe}} = \chi_1$, and $q \leq 2^{n^\epsilon}$ (for LWE security: $\mathsf{LWE}_{n,q,\chi_{\mathsf{lwe}}}$, $\mathsf{LWE\text{-}ss}_{n,q,\chi_{\mathsf{lwe}}}$, $\mathsf{LWE\text{-}sp}_{6n \log q, q, \sigma_{\mathsf{pre}}, \chi_{\mathsf{appr}}}$);
- $m > 2(4\lambda + w) \cdot n \cdot \log q + 12n \cdot \log q$ (for enhanced trapdoor sampling);
- $\sigma > \sqrt{(4\lambda + w) \cdot n \cdot \log q \cdot \log m} + \lambda$ (for enhanced trapdoor sampling);
- $\chi_{\mathsf{big}} = U_{\sigma_{\mathsf{big}}}$ and $\chi_{\mathsf{last}} = U_{\sigma_{\mathsf{last}}}$, where $\sigma_{\mathsf{big}} = \sigma \cdot 2^\lambda, \sigma_{\mathsf{last}} = (m \cdot \sigma)^\ell \cdot 2^\lambda$ (for smudging/security);
- $\sigma_{\mathsf{pre}} = \sigma$, $(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^\ell \cdot (m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}})) \leq q/16$ (for correctness);

- $\sigma_s = \sigma$, $\sqrt{n} \cdot \sigma_s \cdot (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{\ell} \leq q/16$ (for correctness).

First, note that it is not necessary to have distributions $\chi_{\mathsf{lwe}}, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}$ be the same distribution. Keeping all these to be different distributions will only affect the underlying assumptions to which we reduce security. One possible setting of parameters is as follows: $n = (2\lambda \cdot \ell)^{1/\epsilon}$, $m = n^{1+2\epsilon} \cdot w$, $q = 2^{n^{\epsilon}}$, and $\sigma = n \cdot \sqrt{w}$.

We will now describe our Mixed FE construction.

- $\mathsf{Setup}(1^{\lambda}, (1^{k}, 1^{w}, 1^{L})) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ, message length k, branching program width w, and number of times it reads each bit L.[1] It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}$ as described above. Let $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It runs the $\mathsf{EnTrapGen}$ algorithm ℓ times as follows:

$$\forall\ i \in [\ell], \quad (\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^{m}, q).$$

For each $i \in [\ell]$, it interprets matrix $\mathbf{M}_i$ as $4\lambda + w$ matrices with dimensions $n \times m$ arranged as follows:

$$\mathbf{M}_i = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j \in [\lambda], \beta, b \in \{0,1\}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}.$$

[1] Note that here we slightly deviate from our definition as we have 3 separate functionality parameters instead of a single index. This could simply be handled by extending the mixed FE definition to multiple indices.

Also, it samples $4\ell\lambda$ matrices $\{\mathbf{C}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}$ uniformly at random as $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$. Finally, it sets the public parameters and the master secret key as

$$\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}}),$$
$$\mathsf{msk} = \left(\left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}, \{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}, \{T_i\}_{i\in[\ell]}\right).$$

- $\mathsf{KeyGen}(\mathsf{msk}, x \in \{0,1\}^k) \rightarrow \mathsf{sk}$. The key generation algorithm takes as input the master secret key msk and a message $x \in \{0,1\}^k$. Let

$$\mathsf{msk} = \left(\left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}, \{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}, \{T_i\}_{i\in[\ell]}\right).$$

It chooses a secret vector $\widetilde{\mathbf{s}}$ of length n as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors $\mathbf{y}^{(j)}$ of length m as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

The key generation algorithm then chooses $2\ell\lambda$ secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$ and $2(\ell+1)\lambda$ error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ of length n and m, respectively, as

$$\forall\ i \in [\ell], j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ i \in [\ell], j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ j \in [\lambda], \beta \in \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

Let $\widetilde{x} = x^L$, i.e., $\widetilde{x}$ is a $k \cdot L$-bit string obtained by appending string x to itself L times. Next, it computes $2(\ell+1)\lambda$ key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as

follows:

$$\forall\, i \in [\ell+1], j \in [\lambda], \beta \in \{0,1\},$$
$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

Finally, it outputs the secret key sk as

$$\mathsf{sk} = \left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i\in[\ell+1],j\in[\lambda],\beta\in\{0,1\}}\right).$$

- $\mathsf{Enc}(\mathsf{pp}) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$. It first chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and 2ℓ random short matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ as

$$\forall\, i \in [\ell], b \in \{0,1\}, \quad \mathbf{U}_{i,b} \leftarrow \chi_{\mathsf{pre}}^{m\times m}.$$

Finally, it outputs the ciphertext ct as

$$\mathsf{ct} = \left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right).$$

- $\mathsf{SK\text{-}Enc}(\mathsf{msk}, \mathsf{BP}) \to \mathsf{ct}$. The secret key encryption algorithm takes as input the master secret key msk and an input-circling branching program BP. Let

$$\mathsf{msk} = \left(\left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{i\in[\ell],j\in[\lambda],\beta,b\in\{0,1\}}, \{\mathbf{P}_{i,v}\}_{i\in[\ell],v\in[w]}, \{T_i\}_{i\in[\ell]}\right),$$
$$\mathsf{BP} = \left(\{\pi_{i,b} : [w] \to [w]\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc} \in [w], \mathsf{rej} \in [w]\right).$$

It first chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and samples $8\ell\lambda$ matrices $\left\{\mathbf{D}_{i,b}^{(j,\beta)}, \widetilde{\mathbf{D}}_{i,b}^{(j,\beta)}\right\}_{i,j,\beta,b}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$ as follows:

$$\forall\ i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}, \quad \begin{aligned} \mathbf{D}_{i,b}^{(j,\beta)} &= \begin{cases} \mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise,} \end{cases} \\ \widetilde{\mathbf{D}}_{i,b}^{(j,\beta)} &= \begin{cases} \mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases} \end{aligned}$$

In the above cases, we use "$\left(\leftarrow \mathbb{Z}_q^{n\times m}\right)$" to denote the operation of uniformly sampling a dimension $n \times m$ matrix in $\mathbb{Z}_q$. Note that here the sampling is performed *uniformly and independently each time*. Next, the algorithm samples w matrices $\{\mathbf{P}_{\ell+1,v}\}_{v\in[w]}$ for the top level and $2w\ell$ error matrices $\{\mathbf{E}_{i,v}, \widetilde{\mathbf{E}}_{i,v}\}_{i\in[\ell],v\in[w]}$ as follows:

$$\begin{aligned} \forall\ v \in [w], \quad \mathbf{P}_{\ell+1,v} &= \begin{cases} \mathbf{0}^{n\times m} & \text{if } v = \mathsf{rej}, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise,} \end{cases} \\ \forall\ i \in [\ell], v \in [w], \quad \mathbf{E}_{i,v} &\leftarrow \chi_{\mathsf{appr}}^{n\times m}, \widetilde{\mathbf{E}}_{i,v} \leftarrow \chi_{\mathsf{appr}}^{n\times m}. \end{aligned}$$

The algorithm then sets $2w\ell$ matrices $\{\mathbf{Q}_{i,v}, \widetilde{\mathbf{Q}}_{i,v}\}_{i\in[\ell],v\in[w]}$ as follows:

$$\forall\ i \in [\ell], v \in [w], \quad \begin{aligned} \mathbf{Q}_{i,v} &= \mathbf{P}_{i+1,\pi_{i,0}(v)} + \mathbf{E}_{i,v}, \\ \widetilde{\mathbf{Q}}_{i,v} &= \mathbf{P}_{i+1,\pi_{i,1}(v)} + \widetilde{\mathbf{E}}_{i,v}. \end{aligned}$$

Next, for $i \in [\ell]$, we use matrices $\mathbf{M}_i, \mathbf{W}_i, \widetilde{\mathbf{W}}_i$ to represent the following $(4\lambda + w)n \times m$ dimension matrices:

$$\mathbf{M}_i = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \mathbf{W}_i = \begin{bmatrix} \left\{\mathbf{D}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \left\{\mathbf{Q}_{i,v}\right\}_{v\in[w]} \end{bmatrix},$$

$$\widetilde{\mathbf{W}}_i = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_{i,b}^{(j,\beta)}\right\}_{j\in[\lambda],\beta,b\in\{0,1\}} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

Now, the secret key encryption algorithm runs the $\mathsf{EnSamplePre}$ to compute 2ℓ short matrices $\{\mathbf{U}_{i,b}\}_{i,b}$ as

$$\forall\ i \in [\ell], \qquad \begin{array}{l} \mathbf{U}_{i,0} \leftarrow \mathsf{EnSamplePre}(\mathbf{M}_i, T_i, \sigma_{\mathsf{pre}}, \mathbf{W}_i), \\ \mathbf{U}_{i,1} \leftarrow \mathsf{EnSamplePre}(\mathbf{M}_i, T_i, \sigma_{\mathsf{pre}}, \widetilde{\mathbf{W}}_i). \end{array}$$

Finally, it outputs the ciphertext ct as

$$\mathsf{ct} = \left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right).$$

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \{0,1\}$. The decryption algorithm takes as input a secret key sk and a ciphertext ct. Let

$$\mathsf{sk} = \left(x, \left\{\mathbf{t}_i^{(j,\beta)}\right\}_{i\in[\ell+1],j\in[\lambda],\beta\in\{0,1\}}\right), \qquad \mathsf{ct} = \left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right).$$

We will assume the algorithm knows the LWE modulus q (i.e., for instance, the public parameters could be included in the secret keys). Let $\widetilde{x} = x^L$, i.e., $\widetilde{x}$ is a $k \cdot L$-bit string obtained by appending string x to itself L times. The decryption algorithm computes the following:

$$\mathbf{z} = \sum_{j=1}^{\lambda} \sum_{i=1}^{\ell+1} \left(\mathbf{t}_i^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=i}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}\right).$$

Finally, if $\|\mathbf{z}\| \le q/8$, it outputs 0; otherwise it outputs 1.

Theorem 7.2.1. *Assuming the trapdoor scheme $\mathsf{LT}_{\mathsf{en}}$ satisfies the $(q, \sigma_{\mathsf{pre}})$-well-sampledness of the preimage, the $(q, \sigma_{\mathsf{pre}})$-row removal property, the $(q, \chi_{\mathsf{lwe}}, \sigma_{\mathsf{pre}})$-target switching property, assuming that assumptions $\mathsf{LWE}_{n,q,\chi_{\mathsf{lwe}}}$, $\mathsf{LWE\text{-}ss}_{n,q,\chi_{\mathsf{lwe}}}$, and $\mathsf{LWE\text{-}sp}_{6n\log q,\sigma_{\mathsf{pre}},\chi_{\mathsf{appr}}}$ hold (where $n, m, q, \sigma_{\mathsf{pre}}, \chi_{\mathsf{lwe}}, \chi_{\mathsf{appr}}$ are defined as in the construction), for any PPT adversary $\mathcal{A}$ that outputs $(1^k, 1^w, 1^L)$ such that*

the parameter constraints as provided in the construction are satisfied, then there exist negligible functions $negl_1(\cdot), negl_2(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$, $\mathcal{A}$*'s advantage in the* 1*-query restricted function indistinguishability security (see Definition 4.2.2) and* 1*-query restricted accept indistinguishability (see Definition 4.2.4) is at most* $negl_1(\lambda)$ *and* $negl_2(\lambda)$*, respectively.*

Remark 7.2.1 (extending to r-query security). We would like to point out that the above construction can be naturally extended to achieve r-query security for any a priori fixed polynomial r. To understand the modification, we will look ahead to the security proof. Specifically, we will focus on the importance of the λ-bit string tag chosen during encryption. During the proof, we crucially rely on the tag strings tag and tag^* (the first chosen for answering the encryption query, and the second used to answer the challenge query) being distinct at at least one index. Since they are chosen uniformly at random each time, thus we know that $\mathsf{tag} \neq \mathsf{tag}^*$ with probability $1 - \frac{1}{2^\lambda}$. Now if the challenger has to answer r encryption queries instead of just 1, then the modification we consider is to increase the alphabet size of tags such that the tag strings chosen during all encryption queries and the challenge query are distinct at at least one index. (Note that this would also mean that we will have to likewise increase the number of underlying matrices chosen and extend the trapdoor sampling procedure appropriately.) More formally, we will now sample tag strings as a uniformly random $2r^2$-ary string of length-λ (i.e., $\mathsf{tag} \leftarrow \{1, \ldots, 2r^2\}^\lambda$). With this modification we can argue that, with all but negligible probability over the choice of tag strings $\mathsf{tag}_1, \ldots, \mathsf{tag}_r$ and

tag^*, there exists an index $i \leq \lambda$ such that the ith elements in all these tag strings are (pairwise) distinct. With this guarantee, the current proof could be extended to argue r-query security.

7.3 Correctness

We will prove that the mixed FE scheme described above satisfies the correctness property. Our correctness proof is divided into two parts. First, we show that if ct is a mixed FE encryption of branching program BP, then given any secret key sk_x, the decryption algorithm outputs $\mathsf{BP}(x)$ with all-but-negligible probability. Second, we show that if ct is a normal FE ciphertext, then given any secret key sk_x, the decryption algorithm outputs 1 with all-but-negligible probability.

Lemma 7.3.1. *For every $\lambda, k, w, L \in \mathbb{N}$, for every length $k \cdot L$ and width w input-circling branching program BP with input space $\{0,1\}^k$, input $x \in \{0,1\}^k$, the following holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_x, \mathsf{ct}) = \mathsf{BP}(x) : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, (1^k, 1^w, 1^L)); \\ \mathsf{sk}_x \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x); \mathsf{ct} \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, \mathsf{BP})\end{array}\right] \geq 1 - \mathit{negl}_2(\lambda),$$

where $\mathit{negl}_2(\cdot)$ is a negligible function.

Proof. Here and throughout, whenever we say matrices/terms corresponding to (j, β)th strands, then we mean the corresponding matrix/term with the superscript (j, β). Also, when we refer to the matrices/terms along the strands

selected by some tag string tag, then we mean all those matrices/terms with superscripts (j, tag_j) for $j \in [\lambda]$.

Now recall that the setup algorithm chooses matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}, \mathbf{P}_{i,v}$ for $i \in [\ell]$, $j \in [\lambda]$, $\beta, b \in \{0,1\}$, $v \in [w]$. Here all matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{P}_{i,v}$ for any particular value i (i.e., any fixed level) are sampled along with trapdoor information. Now for any input $x \in \{0,1\}^k$, the key generation algorithm chooses vectors $\mathbf{y}^{(j)}, \widetilde{\mathbf{s}}$ such that $\widetilde{\mathbf{s}}$ is short and $\sum_j \mathbf{y}^{(j)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$. It also samples secret vectors $\mathbf{s}_i^{(j,\beta)}$ and error vectors $\mathbf{e}_i^{(j,\beta)}$ and computes the secret key components $\mathbf{t}_i^{(j,\beta)}$ in the special way as described in the construction. Now the mixed FE encryption algorithm samples a λ-bit tag string tag, and it uses the trapdoor information to target $\mathbf{B}_{i,b}^{(j,\beta)}$ matrices to their corresponding $\mathbf{C}_{i,b}^{(j,\beta)}$ matrices only along the strands selected by the tag string tag. Additionally, it also targets the program matrices $\mathbf{P}_{i,v}$ at each level to their counterparts in the next level per the branching program state transition function. For proving correctness we simply show that the final program matrix reached after decryption is either short or random depending upon outcome of the evaluation, and the $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices get canceled at each step, and the error terms are appropriately bounded.

We start by introducing some notation useful for the correctness proof.

- st_i: the state of BP after i steps when evaluated on input x.
- $\widetilde{\mathbf{t}}_i^{(j,\beta)} \stackrel{\text{def}}{=} \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} & \text{if } i = \ell + 1: \end{cases}$

the *error-free* secret key components, i.e., secret key vectors without adding error vectors $\mathbf{e}_i^{(j,\beta)}$.

- $\boldsymbol{\Delta}_i^{(j)} \stackrel{\text{def}}{=} \sum_{\gamma=1}^{i} \mathbf{t}_\gamma^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=\gamma}^{i-1} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}$: the partial sum computed during decryption after using the first i components of the secret key along only the (j,tag_j)th strand.
- $\widetilde{\boldsymbol{\Delta}}_i^{(j)} \stackrel{\text{def}}{=} \begin{cases} \mathbf{s}_i^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)} \cdot \prod_{\alpha=1}^{i-1} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} & \text{if } i \leq \ell, \\ \mathbf{y}^{(j)} \cdot \prod_{\alpha=1}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} & \text{if } i = \ell+1: \end{cases}$ the expected sum during decryption in absence of *errors* after using the first i components of the secret key along the (j,tag_j)th strand.
- $\boldsymbol{\Delta}_i \stackrel{\text{def}}{=} \sum_{j=1}^{\lambda} \boldsymbol{\Delta}_i^{(j)}$ and $\widetilde{\boldsymbol{\Delta}}_i = \sum_{j=1}^{\lambda} \widetilde{\boldsymbol{\Delta}}_i^{(j)}$.
- $\mathbf{err}_i^{(j)} \stackrel{\text{def}}{=} \boldsymbol{\Delta}_i^{(j)} - \widetilde{\boldsymbol{\Delta}}_i^{(j)}$ and $\mathbf{err}_i \stackrel{\text{def}}{=} \boldsymbol{\Delta}_i - \widetilde{\boldsymbol{\Delta}}_i$.
- $\boldsymbol{\Gamma}_i \stackrel{\text{def}}{=} \mathbf{P}_{1,1} \cdot \prod_{\alpha=1}^{i} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}$: the matrix denoting partial branching program evaluation after i decryption steps, i.e., equal to the $\boldsymbol{\Delta}_i$ term ignoring the $\sum_{j=1}^{\lambda} \mathbf{s}_i^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\mathsf{tag}_j)}$ blinding component, and short secret $\widetilde{\mathbf{s}}$ multiplied.

Observe that the decryption algorithm computes $\boldsymbol{\Delta}_{\ell+1}$ and tests whether it is close to zero or not. We start by proving that for all i, j, the error term $\mathbf{err}_i^{(j)}$ is small and bounded. This would help us in arguing that for every i, $\mathbf{err}_i$ is also small, thereby giving us that matrices $\boldsymbol{\Delta}_{\ell+1}$ and $\widetilde{\boldsymbol{\Delta}}_{\ell+1}$ are very close to each other as well. Combining this with the fact that $\widetilde{\boldsymbol{\Delta}}_{\ell+1}$ is either a random matrix or a short matrix, depending upon the output $\mathsf{BP}(x)$, we get that the sum $\boldsymbol{\Delta}_{\ell+1}$ computed by the decryption algorithm is close to zero if $\mathsf{BP}(x) = 0$; otherwise it is a random vector with large entries.

Claim 7.3.1. *There exists a negligible function negl(·) such that*

$$\forall\ i \in [\ell], j \in [\lambda], \quad \left\|\mathbf{err}_i^{(j)}\right\| \leq \left(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}})\right)^i,$$
$$\forall\ j \in [\lambda], \quad \left\|\mathbf{err}_{\ell+1}^{(j)}\right\| \leq \left(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}})\right)^\ell \cdot \left(m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}})\right)$$

with probability $1 - \mathit{negl}(\lambda)$.

Proof. We prove the above claim by inducting on the levels i. Our proof is insensitive to the choice of strand index j; thus for the purposes of this proof, it could be fixed to an arbitrary value.

Base case ($i = 1$) Note that $\mathbf{\Delta}_1^{(j)} = \mathbf{t}_1^{(j,\mathsf{tag}_j)} = \mathbf{s}_1^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\mathsf{tag}_j)}$, where $\mathbf{e}_1^{(j,\mathsf{tag}_j)}$ is a short error vector drawn from χ_{big}^m. Also, we have that $\widetilde{\mathbf{\Delta}}_1^{(j)} = \mathbf{s}_1^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)}$ by definition. Thus, we get that

$$\left\|\mathbf{err}_1^{(j)}\right\| = \left\|\mathbf{\Delta}_1^{(j)} - \widetilde{\mathbf{\Delta}}_1^{(j)}\right\| = \left\|\mathbf{e}_1^{(j,\mathsf{tag}_j)}\right\| \leq \sqrt{m} \cdot \sigma_{\mathsf{big}}$$

with all-but-negligible probability. This completes the proof of the base case. For the induction step, we assume that the above claim holds for i^* and show that it holds for $i^* + 1$ as well.

Induction step We know that $\mathbf{\Delta}_{i^*+1}^{(j)} = \mathbf{\Delta}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}$. Since $\mathbf{\Delta}_{i^*}^{(j)} = \widetilde{\mathbf{\Delta}}_{i^*}^{(j)} + \mathbf{err}_{i^*}^{(j)}$, we get that

$$\begin{aligned}\mathbf{\Delta}_{i^*+1}^{(j)} &= (\widetilde{\mathbf{\Delta}}_{i^*}^{(j)} + \mathbf{err}_{i^*}^{(j)}) \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \mathbf{t}_{i^*+1}^{(j,\mathsf{tag}_j)} \\ &= \widetilde{\mathbf{\Delta}}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \mathbf{t}_{i^*+1}^{(j,\mathsf{tag}_j)} + \mathbf{err}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}.\end{aligned}$$

Now, from our construction we have that

$$\begin{aligned}
&\mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} = \mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{C}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} \\
&\Rightarrow \mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{t}}_{i^*+1}^{(j,\mathsf{tag}_j)} = \mathbf{s}_{i^*+1}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\mathsf{tag}_j)} \\
&\Rightarrow \widetilde{\mathbf{\Delta}}^{(j)}_{i^*} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{t}}_{i^*+1}^{(j,\mathsf{tag}_j)} = \widetilde{\mathbf{\Delta}}^{(j)}_{i^*+1}.
\end{aligned}$$

Combining the fact that $\mathbf{t}_{i^*+1}^{(j,\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_{i^*+1}^{(j,\mathsf{tag}_j)} + \mathbf{e}_{i^*+1}^{(j,\mathsf{tag}_j)}$ with the above equations, we get that, with all-but-negligible probability, the following holds:

$$\begin{aligned}
\mathbf{err}_{i^*+1}^{(j)} = \mathbf{\Delta}_{i^*+1}^{(j)} - \widetilde{\mathbf{\Delta}}^{(j)}_{i^*+1} &= \mathbf{e}_{i^*+1}^{(j,\mathsf{tag}_j)} + \mathbf{err}_{i^*}^{(j)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} \\
\Rightarrow \left\|\mathbf{err}_{i^*+1}^{(j)}\right\| &\leq \left\|\mathbf{e}_{i^*+1}^{(j,\mathsf{tag}_j)}\right\| + \left\|\mathbf{err}_{i^*}^{(j)}\right\| \cdot \|\mathbf{U}_{i^*,\widetilde{x}_{i^*}}\| \\
&\leq \sqrt{m} \cdot \sigma^* + \left(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}})\right)^{i^*} \cdot (m \cdot \sigma_{\mathsf{pre}}) \\
&\leq \left(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}})\right)^{i^*} \cdot \left(m \cdot (\sigma^* + \sigma_{\mathsf{pre}})\right),
\end{aligned}$$

where $\sigma^* = \sigma_{\mathsf{big}}$ if $i^* < \ell$; otherwise $\sigma^* = \sigma_{\mathsf{last}}$. This completes the proof of the above claim. ■

From the above claim, we get that for every $j \in [\lambda]$, $\left\|\mathbf{err}_{\ell+1}^{(j)}\right\| \leq (m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}}))^{\ell} \cdot (m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}}))$. Thus, by the triangle inequality we can claim that (with all-but-negligible probability)

$$\|\mathbf{err}_{\ell+1}\| \leq \lambda \cdot \left(m \cdot (\sigma_{\mathsf{big}} + \sigma_{\mathsf{pre}})\right)^{\ell} \cdot \left(m \cdot (\sigma_{\mathsf{last}} + \sigma_{\mathsf{pre}})\right) \leq q/16.$$

Next, we show that $\mathbf{\Gamma}_i$ is close to $\mathbf{P}_{i+1,\mathsf{st}_i}$. In other words, the partial branching program evaluation is correct.

Claim 7.3.2. *For all $i \in \{0, \ldots, \ell\}$, $\|\mathbf{\Gamma}_i - \mathbf{P}_{i+1,\mathsf{st}_i}\| \leq \left(m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}})\right)^i$ with probability $1 - \mathit{negl}(\lambda)$, where $\mathit{negl}(\cdot)$ is a negligible function.*

Proof. We prove the above claim by inducting on the levels i.

Base case ($i = 0$) Note that $\mathbf{\Gamma}_0$ is simply equal to $\mathbf{P}_{1,1}$ as starting state $\mathsf{st}_0 = 1$. Thus, we get that

$$\|\mathbf{\Gamma}_0 - \mathbf{P}_{1,\mathsf{st}_0}\| = 0.$$

This completes the proof of the base case. For the induction step, we assume that the above lemma holds for $i^* - 1$ and show that it holds for i^* as well.

Induction step We know that $\mathbf{\Gamma}_{i^*} = \mathbf{\Gamma}_{i^*-1} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}$. Recall that, per our construction, $\mathbf{U}_{i^*,\widetilde{x}_{i^*}}$ targets $\mathbf{P}_{i^*,\mathsf{st}_{i^*-1}}$ to $\mathbf{P}_{i^*+1,\mathsf{st}_{i^*}} + \mathbf{Err}_1$, where $\mathbf{Err}_1$ is an $n \times m$ matrix sampled uniformly from $\chi^{n\times m}_{\mathsf{appr}}$. Concretely, this gives that

$$\mathbf{P}_{i^*,\mathsf{st}_{i^*-1}} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} = \mathbf{P}_{i^*+1,\mathsf{st}_{i^*}} + \mathbf{Err}_1, \text{ where } \|\mathbf{Err}_1\| \leq m \cdot \sigma_{\mathsf{appr}}.$$

By our inductive hypothesis, we have that $\mathbf{\Gamma}_{i^*-1} = \mathbf{P}_{i^*,\mathsf{st}_{i^*-1}} + \mathbf{Err}_2$, where $\|\mathbf{Err}_2\| \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{i^*-1}$. Thus, we can rewrite matrix $\mathbf{\Gamma}_{i^*}$ as follows:

$$\begin{aligned}\mathbf{\Gamma}_{i^*} &= \left(\mathbf{P}_{i^*,\mathsf{st}_{i^*-1}} + \mathbf{Err}_2\right) \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}} \\ &= \mathbf{P}_{i^*+1,\mathsf{st}_{i^*}} + \left(\mathbf{Err}_1 + \mathbf{Err}_2 \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}\right).\end{aligned}$$

Now we have that

$$\begin{aligned}\|\mathbf{Err}_1 + \mathbf{Err}_2 \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}\| &\leq m \cdot \sigma_{\mathsf{appr}} + (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{i^*-1} \cdot m \cdot \sigma_{\mathsf{pre}} \\ &\leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^{i^*}.\end{aligned}$$

Thus, the claim follows. ∎

From the above claim, we get that (with all-but-negligible probability) $\|\mathbf{\Gamma}_\ell - \mathbf{P}_{\ell+1,\mathsf{st}_\ell}\| \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^\ell$. Next, we show that $\widetilde{\mathbf{\Delta}}_{\ell+1}$ has low norm if the output of branching program is 0; otherwise it is not upper-bounded with all-but-negligible probability.

Claim 7.3.3. *There exists a negligible function negl$(\cdot)$ such that*

$$\left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| = \begin{cases} \leq q/16 & \text{if } \mathsf{BP}(x) = 0, \\ \geq q/4 & \text{if } \mathsf{BP}(x) = 1 \end{cases}$$

with probability $1 - \text{negl}(\lambda)$.

Proof. We know that $\widetilde{\mathbf{\Delta}}_{\ell+1}$ could be written as follows:

$$\widetilde{\mathbf{\Delta}}_{\ell+1} = \left(\sum_{j=1}^{\lambda} \mathbf{y}^{(j)}\right) \cdot \prod_{\alpha=1}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}.$$

Since $\left(\sum_{j=1}^{\lambda} \mathbf{y}^{(j)}\right) = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$, this gives that $\widetilde{\mathbf{\Delta}}_{\ell+1} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} \cdot \prod_{\alpha=1}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha}$ $= \widetilde{\mathbf{s}} \cdot \mathbf{\Gamma}_\ell$. Using Claim 7.3.2, we get that $\mathbf{\Gamma}_\ell = \mathbf{P}_{\ell+1,\mathsf{st}_\ell} + \mathbf{Err}$, where $\|\mathbf{Err}\| \leq (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^\ell$. Also, we know that $\mathbf{P}_{\ell+1,\mathsf{rej}} = \mathbf{0}^{n \times m}$, and $\mathbf{P}_{\ell+1,\mathsf{acc}}$ is a uniformly random $n \times m$ matrix. Thus, we get that with all-but-negligible probability

$$\begin{aligned} \mathsf{BP}(x) = 0 \Rightarrow \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| = \|\widetilde{\mathbf{s}} \cdot \mathbf{Err}\| &\leq \|\widetilde{\mathbf{s}}\| \cdot (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^\ell \\ &\leq \sqrt{n} \cdot \sigma_s \cdot (m \cdot (\sigma_{\mathsf{pre}} + \sigma_{\mathsf{appr}}))^\ell \leq q/16, \end{aligned}$$

$$\mathsf{BP}(x) = 1 \Rightarrow \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| = \|\widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{acc}} + \widetilde{\mathbf{s}} \cdot \mathbf{Err}\| \geq \|\widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{acc}}\| - q/16 \geq q/4,$$

where the last inequality follows from the fact that $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{acc}}$ is a uniformly random vector. This completes the proof of the above claim. ∎

By the triangle inequality, we know that

$$\left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| - \|\mathbf{err}_{\ell+1}\| \leq \|\mathbf{\Delta}_{\ell+1}\| \leq \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| + \|\mathbf{err}_{\ell+1}\|.$$

Combining this with the above claims, we can conclude that with all-but-negligible probability

$$\begin{aligned}
\mathsf{BP}(x) = 0 &\Rightarrow \|\mathbf{\Delta}_{\ell+1}\| \leq q/16 + q/16 \leq q/8, \\
\mathsf{BP}(x) = 1 &\Rightarrow \left\|\widetilde{\mathbf{\Delta}}_{\ell+1}\right\| \geq q/4 - q/16 > q/8.
\end{aligned}$$

Thus, for any input x and branching program BP, the mixed FE encryption algorithm is correct with all-but-negligible probability. This concludes the proof of Lemma 7.3.1. ■

Lemma 7.3.2. *For every $\lambda, k, w, L \in \mathbb{N}$, for every length $k \cdot L$ and width w input-circling branching program BP with input space $\{0,1\}^k$, input $x \in \{0,1\}^k$, the following holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_x, \mathsf{ct}) = 1 : \begin{array}{c}(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, (1^k, 1^w, 1^L)); \\ \mathsf{sk}_x \leftarrow \mathsf{KeyGen}(\mathsf{msk}, x); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp})\end{array}\right] \geq 1 - \mathit{negl}_1(\lambda),$$

where $\mathit{negl}_1(\cdot)$ is a negligible function.

Proof. Recall that the output of a normal encryption algorithm is simply independently drawn 2ℓ short Gaussian matrices $\{\mathbf{U}_{i,b}\}$. Now the decryption algorithm performs the computation

$$\mathbf{z} = \sum_{j=1}^{\lambda} \sum_{i=1}^{\ell+1} \left(\mathbf{t}_i^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=i}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} \right).$$

Here we could rewrite $\mathbf{z}$ as

$$\mathbf{z} = \sum_{j=1}^{\lambda} \left(\sum_{i=1}^{\ell-1} \left(\mathbf{t}_i^{(j,\mathsf{tag}_j)} \cdot \prod_{\alpha=i}^{\ell} \mathbf{U}_{\alpha,\widetilde{x}_\alpha} \right) + \mathbf{t}_\ell^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} + \mathbf{t}_{\ell+1}^{(j,\mathsf{tag}_j)} \right).$$

From our construction we know that for any j,

$$\begin{aligned} \mathbf{t}_\ell^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} + \mathbf{t}_{\ell+1}^{(j,\mathsf{tag}_j)} = & -\mathbf{s}_{\ell-1}^{(j,\mathsf{tag}_j)} \cdot \mathbf{C}_{\ell-1,\widetilde{x}_{\ell-1}}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} + \mathbf{e}_\ell^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} + \mathbf{e}_{\ell+1}^{(j,\mathsf{tag}_j)} \\ & + \mathbf{s}_\ell^{(j,\mathsf{tag}_j)} \cdot \left(\mathbf{B}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} - \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \right). \end{aligned}$$

Now note that since $\mathbf{U}_{\ell,\widetilde{x}_\ell}$ is sampled independently from $\chi_{\mathsf{pre}}^{m\times m}$ and $\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)}$ is a uniform $n \times m$ matrix, thus the matrix $\mathbf{B}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} - \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)}$ is also a uniformly random matrix. Also, secret vector $\mathbf{s}_\ell^{(j,\mathsf{tag}_j)}$ is a length n random vector, and thus the component $\mathbf{s}_\ell^{(j,\mathsf{tag}_j)} \cdot \left(\mathbf{B}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \cdot \mathbf{U}_{\ell,\widetilde{x}_\ell} - \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)} \right)$ is a random vector as well. Now since this is *independent* of all other components as neither $\mathbf{s}_\ell^{(j,\mathsf{tag}_j)}$ nor $\mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\mathsf{tag}_j)}$ appears in any other term in sum vector $\mathbf{z}$, thus the distribution of $\mathbf{z}$ is that of a uniformly random vector over the choice of coins used during setup, key generation, and encryption. Since we know that the ℓ_2-norm of a random vector in $\mathbb{Z}_q^m$ is at least $q/8$ with all-but-negligible probability, therefore the claim follows. ■

7.4 Security proof

We now prove that the mixed FE scheme described in Section 7.2 satisfies the 1-query restricted function indistinguishability as well as the 1-query restricted accept indistinguishability security properties. Our proof is divided into two components where we first prove function indistinguishability, and

later prove accept indistinguishability. Both proofs proceed via a sequence of hybrid games.

7.4.1 1-query restricted function indistinguishability

Below we provide a sequence of hybrid games that we later use to argue function indistinguishability security.

Game 0 This corresponds to the original 1-query restricted function indistinguishability security game.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$.
  2. Next, it samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
  $$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$
  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$ and a λ-bit string $\mathsf{tag}^* \leftarrow \{0,1\}^\lambda$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{ \pi_{i,b}^{(\gamma)} : [w] \to [w] \right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as

$$\forall\, \alpha \in [\ell], \left(\left\{ \mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1} \right\} \right) \leftarrow \mathsf{Mixed\text{-}SubEnc} \left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{ \mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)} \right\}_{(i,j,\beta,b) \in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \\ \{T_i\}_{i \in [\ell]}, \mathsf{BP}^{(\gamma)} \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{ \mathbf{U}^*_{i,b} \right\}_{i \in [\ell], b \in \{0,1\}} \right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and responds as follows:

     (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc} \left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{ \mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)} \right\}_{(i,j,\beta,b) \in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \\ \{T_i\}_{i \in [\ell]}, \mathsf{BP} \end{array} \right).$$

Mixed-SubEnc

Inputs:

- Tag tag, Level α, Set $S \subseteq [\ell] \times [\lambda] \times \{0,1\}^2$, Matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\}_{(i,j,\beta,b)\in S}$, $\{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}$, Trapdoors $\{T_i\}_{i\in[\ell]}$;
- $\mathsf{BP} = \big(\{\pi_{i,b} : [w] \to [w]\}_{(i,b)\in[\ell]\times\{0,1\}}, \mathsf{acc} \in [w], \mathsf{rej} \in [w]\big)$.

Output: Matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$.

Execution: Let S_α denote the following set:

$$S_\alpha = \big\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \text{ such that } (\alpha, j, \beta, b) \in S\big\}.$$

Sample matrices $\{\mathbf{D}_b^{(j,\beta)}, \widetilde{\mathbf{D}}_b^{(j,\beta)}\}_{(j,\beta,b)\in S_\alpha}$ as

$$\forall\ (j,\beta,b) \in S_\alpha, \quad \mathbf{D}_b^{(j,\beta)} = \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0, \\ \big(\leftarrow \mathbb{Z}_q^{n\times m}\big) & \text{otherwise,} \end{cases}$$
$$\widetilde{\mathbf{D}}_b^{(j,\beta)} = \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1, \\ \big(\leftarrow \mathbb{Z}_q^{n\times m}\big) & \text{otherwise.} \end{cases}$$

Sample $2w$ error matrices as $\mathbf{E}_v \leftarrow \chi_{\mathsf{appr}}^{n\times m}, \widetilde{\mathbf{E}}_v \leftarrow \chi_{\mathsf{appr}}^{n\times m}$ for $v \in [w]$. Also, if $\alpha = \ell$, sample w matrices $\{\mathbf{P}_{\ell+1,v}\}_{v\in[w]}$ for the top level as

$$\forall\ v \in [w], \quad \mathbf{P}_{\ell+1,v} = \begin{cases} \mathbf{0}^{n\times m} & \text{if } v = \mathsf{rej}, \\ \big(\leftarrow \mathbb{Z}_q^{n\times m}\big) & \text{otherwise.} \end{cases}$$

Next, set $2w$ matrices $\left\{\mathbf{Q}_v, \widetilde{\mathbf{Q}}_v\right\}_{v\in[w]}$ as

$$\forall\ v \in [w], \quad \begin{matrix} \mathbf{Q}_v = \mathbf{P}_{\alpha+1,\pi_{\alpha,0}(v)} + \mathbf{E}_v, \\ \widetilde{\mathbf{Q}}_v = \mathbf{P}_{\alpha+1,\pi_{\alpha,1}(v)} + \widetilde{\mathbf{E}}_v. \end{matrix}$$

(Execution continues in Fig. 7.2.)

Figure 7.1: Routine Mixed-SubEnc

(b) Finally, it sends the ciphertext as $\big(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\big)$.

Mixed-SubEnc **(continued)**

Let matrices $\mathbf{M}, \mathbf{W}, \widetilde{\mathbf{W}}$ represent the following $(|S_\alpha| + w)n \times m$ dimension matrices:

$$\mathbf{M} = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \left\{\mathbf{D}_{b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{Q}_{i,v}\}_{v\in[w]} \end{bmatrix},$$

$$\widetilde{\mathbf{W}} = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_{b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

Run the $\mathsf{EnSamplePre}$ to compute matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$ as

$$\mathbf{U}_0 \leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \mathbf{W}),$$
$$\mathbf{U}_1 \leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \widetilde{\mathbf{W}}).$$

Figure 7.2: Routine Mixed-SubEnc

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

   $$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

   (b) It then chooses secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$ and error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ as

   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
   $$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\},$$

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)}, & i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}, & 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}, & i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game 1 This is identical to the previous game, except the challenger now chooses both tags tag^* and tag at the beginning during the setup phase, and it aborts if $\mathsf{tag}^* = \mathsf{tag}$.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.

  2. Next, it samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)}\end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

(a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

$$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

(b) It then chooses secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$, error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ as

$$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m. \end{aligned}$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\},$$

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)}, & i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}, & 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}, & i = \ell + 1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Notation In all the following hybrid games, let j^* denote the smallest index in $\{1, \ldots, \lambda\}$ such that $\mathsf{tag}^*_{j^*} \neq \mathsf{tag}_{j^*}$, i.e., $j^* = \min\big\{j \in [\lambda] \ : \ \mathsf{tag}^*_j \neq \mathsf{tag}_j\big\}$. Since the challenger aborts whenever $\mathsf{tag}^* = \mathsf{tag}$, thus j^* always exists whenever the challenger does not abort. Additionally, let $\beta^* = \mathsf{tag}^*_{j^*}$.

Game 2 This is identical to the previous game, except the challenger, while answering a secret key query, now puts the $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$ component in $\mathbf{y}^{(j^*)}$ instead of $\mathbf{y}^{(\lambda)}$, and the rest are sampled uniformly at random.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.

2. Next, it samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{ \pi_{i,b}^{(\gamma)} : [w] \to [w] \right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as

$$\forall\, \alpha \in [\ell], \quad \left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc} \left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

   $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$. Next, it sets vector $\mathbf{y}^{(j^*)}$ as

   $$\mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)}.$$

   (b) It then chooses secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$, error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ as

   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
   $$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\},$$
$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)}, & i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}, & 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}, & i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Next, we have a sequence of 4ℓ hybrid experiments, $\mathsf{Game}\ 3.i^*.\{1,2,3,4\}$ for $i^* = 1$ to ℓ.

$\underline{\mathsf{Game}\ 3.i^*.1}$ In hybrids $\mathsf{Game}\ 3.i^*.1$, the $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices for the j^*th strands and levels $i < i^*$ are *not* sampled (at all) along with other level i matrices (i.e., (j^*, β^*) and $(j^*, 1-\beta^*)$ strands); ciphertext components for levels $i < i^*$ are used to target only the remaining matrices; i.e., the ciphertext matrices do not target $\mathbf{B}_{i,b}^{(j,\beta)}$ matrices for $j = j^*$ and $i < i^*$ to some prespecified $\mathbf{C}_{i,b}^{(j,\beta)}$ or random matrices. Also, the first $i^* - 1$ components in each secret key are set to be uniformly random vectors, and the next component is hardwired such that correctness holds, and also some smudgeable noise is introduced into these components. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i < i^*, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$
$$\forall\ i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

Also, let

$$\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^*, \\ \widetilde{n} & \text{for } i \geq i^*. \end{cases}$$

Set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ :\ (j,\beta,b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$

$S^* = \hat{S}$.

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$\left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \\ \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \left\{\mathbf{U}_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after $i^* - 1$ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} \text{ if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} \text{ if } i = \ell + 1, \end{cases}$$

$$\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)},$$

$$\forall\ \beta \in \{0,1\},\quad \mathbf{t}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)},$$

$$\forall\ (i,\beta) \in ([\ell+1] \setminus [i^*]) \times \{0,1\},$$

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} \text{ if } i \le \ell, \\ -\mathbf{s}_{\ell}^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} \text{ if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game $3.i^*.2$ This is identical to the previous game, except the (i^*+1)th key component in the j^*th strands is also hardwired. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i < i^*,\quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$

$$\forall\ i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

Also, let

$$\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^*, \\ \widetilde{n} & \text{for } i \geq i^*. \end{cases}$$

Set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

     $$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m, \end{aligned}$$

$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after $i^* - 1$ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

Also, for $\beta \in \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j^*,\beta)} = \mathbf{0}^{n \times m}$, and let $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}$ denote the following vector:

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}$$

$$+ \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)},$$

$$\forall\ (i,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \{0,1\},$$

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} \text{ if } i \le \ell, \\ -\mathbf{s}_\ell^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} \text{ if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game $3.i^*.3$ This is identical to the previous game, except $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for strands $j = j^*$ and levels $i = i^*$ are *not* sampled along with other level i^* matrices, but instead they are sampled uniformly at random. Also, ciphertext components for level i^* are used to target only the remaining matrices. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i < i^*+1, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$

$$\forall\ i \geq i^* + 1, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

Also, let

$$\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^* + 1, \\ \widetilde{n} & \text{for } i \geq i^* + 1. \end{cases}$$

Set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$

$$\forall\ (\beta, b) \in \{0,1\}^2, \quad \mathbf{B}_{i^*,b}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i, j, \beta, b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$

$S^* = \hat{S}$.

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

     $$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m, \end{aligned}$$

$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after $i^* - 1$ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

Also, for $\beta \in \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j^*,\beta)} = \mathbf{0}^{n\times m}$, and let $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)}$ denote the following vector:

$$\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,\beta) \in [i^*-1] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)}.$$

For all $\beta \in \{0,1\}$,

$$\mathbf{t}_{i^*}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)},$$

$$\mathbf{t}_{i^*+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)},$$

$\forall\ (i,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \{0,1\}$,

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} \text{ if } i \le \ell, \\ -\mathbf{s}_\ell^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} \text{ if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game $3.i^*.4$ This is identical to the previous game, except the i^*th level key component in the j^*th strands is a uniformly random n length vector; i.e., all first i^* level components in the j^*th strand are random elements. Also, we no longer sample the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for strands $j = j^*$ and levels $i = i^*$ at all. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the

adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i < i^* + 1, \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2,$$
$$\forall\ i \geq i^* + 1, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

Also, let

$$\widetilde{n}_i = \begin{cases} \widetilde{n} - 4n & \text{for } i < i^* + 1, \\ \widetilde{n} & \text{for } i \geq i^* + 1. \end{cases}$$

Set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1)

as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$\left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,$$
$$\forall\ (i,\beta) \in [i^*] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m,$$
$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after $i^* - 1$ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

Also, for $\beta \in \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j^*,\beta)} = \mathbf{0}^{n \times m}$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,\beta) \in [i^*] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}$$

$$+ \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)},$$

$\forall\ (i,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \{0,1\}$,

$$\mathbf{t}_i^{(j^*,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j^*,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j^*,\beta)} + \mathbf{s}_i^{(j^*,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)} \text{ if } i \leq \ell, \\ -\mathbf{s}_\ell^{(j^*,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} \text{ if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game 4 This is identical to the previous game, i.e., Game $3.\ell.4$. For ease of exposition, we describe it in detail below.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i \in [\ell], \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2.$$

  Also, let $\widetilde{n}_i = \widetilde{n} - 4n$ for all $i \in [\ell]$.

  Set $\hat{S} = \big\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ :\ (j,\beta,b) \in S^{(i)}\big\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^{(\gamma)} \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

(a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,\beta) \in [\ell]\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after ℓ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1]\times([\lambda]\setminus\{j^*\})\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}^{(\beta)}_{i,b} = \begin{cases} \mathbf{U}^*_{i,b} & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}^{(\beta^*)}_{\ell+1,v}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}^{(1-\beta^*)}_{\ell+1,v}$ be the top level matrices chosen while computing the query ciphertext. Next, it computes key vectors $\{\mathbf{t}^{(j,\beta)}_i\}_{i,j,\beta}$ as follows. For all $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}^{(j,\beta)}_i = \begin{cases} \mathbf{s}^{(j,\beta)}_1 \cdot \mathbf{B}^{(j,\beta)}_{1,\widetilde{x}_1} + \mathbf{y}^{(j)} + \mathbf{e}^{(j,\beta)}_1 & \text{if } i = 1, \\ -\mathbf{s}^{(j,\beta)}_{i-1} \cdot \mathbf{C}^{(j,\beta)}_{i-1,\widetilde{x}_{i-1}} + \mathbf{s}^{(j,\beta)}_i \cdot \mathbf{B}^{(j,\beta)}_{i,\widetilde{x}_i} + \mathbf{e}^{(j,\beta)}_i & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}^{(j,\beta)}_\ell \cdot \mathbf{C}^{(j,\beta)}_{\ell,\widetilde{x}_\ell} + \mathbf{e}^{(j,\beta)}_{\ell+1} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \mathbf{t}^{(j^*,\beta)}_i = \widetilde{\mathbf{t}}^{(j^*,\beta)}_i + \mathbf{e}^{(j^*,\beta)}_i.$$

For all $\beta \in \{0,1\}$,

$$\mathbf{t}^{(j^*,\beta)}_{\ell+1} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}^{(j^*,\beta)}_\alpha \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}} + \mathbf{e}^{(j^*,\beta)}_{\ell+1}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}^{(j,\beta)}_i\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Next, we have a sequence of ℓ hybrid experiments, Game $4.i^*$ for $i^* = 1$ to ℓ.

Game $4.i^*$ This is identical to the previous game, except the challenger uses the routine Mixed-SubEnc* to generate the first i^* components of both the challenge

as well as the query ciphertext. This routine is similar to Mixed-SubEnc, except that Mixed-SubEnc* outputs ciphertext components that map the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ matrices to uniformly random matrices (instead of mapping to $\{\mathbf{P}_{i^*+1,\pi(v)}\}_{v\in[w]}$ as in Mixed-SubEnc).

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i \in [\ell], \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2.$$

  Also, let $\widetilde{n}_i = \widetilde{n} - 4n$ for all $i \in [\ell]$.

  Set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ :\ (j,\beta,b) \in S^{(i)}\right\}$.

  2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{ \pi_{i,b}^{(\gamma)} : [w] \to [w] \right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w] \right),$$
$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ and $\mathsf{Mixed\text{-}SubEnc}^*$ routines (described in Figs. 7.1 and 7.3) as

$$\forall\ \alpha \in [i^*],$$
$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}^* \left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \\ \{T_i\}_{i \in [\ell]} \end{array} \right),$$
$$\forall\ \alpha \in [\ell] \setminus [i^*],$$
$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc} \left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \\ \{T_i\}_{i \in [\ell]}, \mathsf{BP}^{(\gamma)} \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

Mixed-SubEnc*

Inputs:

- Tag tag, Level α, Set $S \subseteq [\ell] \times [\lambda] \times \{0,1\}^2$, Matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\}_{(i,j,\beta,b)\in S}$, $\{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}$, Trapdoors $\{T_i\}_{i\in[\ell]}$.

Output: Matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$.

Execution: Let S_α denote the following set:

$$S_\alpha = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2 \text{ such that } (\alpha,j,\beta,b) \in S\right\}.$$

Sample matrices $\{\mathbf{D}_b^{(j,\beta)}, \widetilde{\mathbf{D}}_b^{(j,\beta)}\}_{(j,\beta,b)\in S_\alpha}$ as

$$\forall\ (j,\beta,b) \in S_\alpha, \quad \begin{aligned} \mathbf{D}_b^{(j,\beta)} &= \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise,} \end{cases} \\ \widetilde{\mathbf{D}}_b^{(j,\beta)} &= \begin{cases} \mathbf{C}_{\alpha,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases} \end{aligned}$$

Sample $2w$ matrices matrices $\left\{\mathbf{Q}_v, \widetilde{\mathbf{Q}}_v\right\}_{v\in[w]}$ as

$$\forall\ v \in [w], \quad \begin{aligned} \mathbf{Q}_v &\leftarrow \mathbb{Z}_q^{n\times m}, \\ \widetilde{\mathbf{Q}}_v &\leftarrow \mathbb{Z}_q^{n\times m}. \end{aligned}$$

(Execution continues in Fig. 7.4.)

Figure 7.3: Routine Mixed-SubEnc*

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc and Mixed-SubEnc* routines (described in Figs. 7.1 and 7.3) as

   $\forall\ \alpha \in [i^*],$

Mixed-SubEnc* **(continued)**

Let matrices $\mathbf{M}, \mathbf{W}, \widetilde{\mathbf{W}}$ represent the following $(|S_\alpha| + w)n \times m$ dimension matrices:

$$\mathbf{M} = \begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \left\{\mathbf{D}_{b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \{\mathbf{Q}_{i,v}\}_{v\in[w]} \end{bmatrix},$$

$$\widetilde{\mathbf{W}} = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_{b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

Run the EnSamplePre to compute matrices $\{\mathbf{U}_0, \mathbf{U}_1\}$ as

$$\mathbf{U}_0 \leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \mathbf{W}),$$
$$\mathbf{U}_1 \leftarrow \mathsf{EnSamplePre}(\mathbf{M}, T_\alpha, \sigma_{\mathsf{pre}}, \widetilde{\mathbf{W}}).$$

Figure 7.4: Routine Mixed-SubEnc*

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \text{Mixed-SubEnc}^*\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]} \end{array}\right),$$

$\forall\ \alpha \in [\ell] \setminus [i^*]$,

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \text{Mixed-SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after ℓ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing the query ciphertext.[2] Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows.

[2]Technically, in $\mathsf{Game}\ 4.(\ell+1)$, $\mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta)}$ is not sampled during $\mathsf{Mixed\text{-}SubEnc}^*$. For that experiment, we will assume these matrices are chosen for the first key query and used for all remaining keys.

For all tuples $(i, j, \beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1, \end{cases}$$

$$\forall\ (i, \beta) \in [\ell] \times \{0, 1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)},$$

$$\forall\ \beta \in \{0, 1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

Game 5 This is identical to the previous game, i.e., Game $4.\ell$. For ease of exposition, we describe it in detail below.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and descriptions of two branching programs $(\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)})$ to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0, 1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the

adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i \in [\ell], \quad S^{(i)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2.$$

Also, let $\widetilde{n}_i = \widetilde{n} - 4n$ for all $i \in [\ell]$.

Set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ :\ (j,\beta,b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random bit $\gamma \leftarrow \{0,1\}$. Let

$$\mathsf{BP}^{(\gamma)} = \left(\left\{\pi_{i,b}^{(\gamma)} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^{(\gamma)} \in [w], \mathsf{rej}^{(\gamma)} \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}^*$ routine (described in Fig. 7.3) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}^*\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]} \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}^*$ routine (described in Fig. 7.3) as follows. For all $\alpha \in [\ell]$,

     $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}^* \left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]} \end{array} \right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}$ as follows:

     $$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m. \end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, $\mathsf{BP}^{(\gamma)}$ after ℓ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times ([\lambda] \setminus \{j^*\}) \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

For the first key query, it samples matrices $\mathbf{P}_{\ell+1,v}^{(\beta)}$ for $v \in [w], \beta \in \{0,1\}$ as follows:[3]

$$\mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta)} = \begin{cases} \mathbf{0}^{n \times m} & \text{if } v = \mathsf{rej}, \\ \left(\leftarrow \mathbb{Z}_q^{n \times m}\right) & \text{otherwise.} \end{cases}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,\beta) \in [\ell] \times \{0,1\}, \quad \mathbf{t}_i^{(j^*,\beta)} = \widetilde{\mathbf{t}}_i^{(j^*,\beta)} + \mathbf{e}_i^{(j^*,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ', and wins if $\gamma' = \gamma$.

[3] Recall, as defined in $\mathsf{Game}\ 4.(\ell+1)$, that these matrices are sampled only for the first key query, and all remaining key queries use the same matrices.

7.4.2 Indistinguishability of hybrid games in Section 7.4.1

We will now show that the hybrid experiments described above are computationally indistinguishable. For any PPT adversary $\mathcal{A}$, let $\mathsf{Adv}_{\mathcal{A},x}(\cdot)$ denote the advantage of $\mathcal{A}$ in $\mathsf{Game}\ x$.

Lemma 7.4.1. *There exists a negligible function $negl(\cdot)$ such that for any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},0}(\lambda) - \mathsf{Adv}_{\mathcal{A},1}(\lambda) \leq negl(\lambda)$.*

Proof. The only difference between $\mathsf{Game}\ 0$ and $\mathsf{Game}\ 1$ is that the challenger aborts if $\mathsf{tag}^* = \mathsf{tag}$. The probability of this event is $2^{-\lambda}$, and it is independent of the adversary's choice of (k, w, L) and $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$ in the setup phase. As a result, for any adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A},0}(\lambda) - \mathsf{Adv}_{\mathcal{A},1}(\lambda) \leq 2^{-\lambda}$. ■

Lemma 7.4.2. *For any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},1}(\lambda) = \mathsf{Adv}_{\mathcal{A},2}(\lambda)$.*

Proof. The only difference between the two hybrids is with respect to the keys. In $\mathsf{Game}\ 1$, for each key query, the challenger chooses $\lambda - 1$ uniformly random vectors $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j < \lambda$ and sets $\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j \in [\lambda-1]} \mathbf{y}^{(j)}$. In $\mathsf{Game}\ 2$, the challenger chooses $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$ and sets $\mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)}$. Fix all $\mathbf{y}^{(j)}$ for $j \notin \{j^*, \lambda\}$ and $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$. Then the following two distributions are identical:

$$\left\{ \left(\mathbf{y}^{(j^*)}, \mathbf{y}^{(\lambda)}\right) : \begin{array}{l} \mathbf{y}^{(j^*)} \leftarrow \mathbb{Z}_q^m; \\ \mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j \in [\lambda-1]} \mathbf{y}^{(j)} \end{array} \right\},$$

$$\left\{ \left(\mathbf{y}^{(j^*)}, \mathbf{y}^{(\lambda)}\right) : \begin{array}{l} \mathbf{y}^{(\lambda)} \leftarrow \mathbb{Z}_q^m; \\ \mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)} \end{array} \right\}.$$

This implies that the distributions in $\mathsf{Game}\ 1$ and $\mathsf{Game}\ 2$ are identical. ■

Lemma 7.4.3. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq negl(\lambda)$.*

Proof. Let us first consider the differences between Game 2 and Game 3.1.1. The setup and challenge phases are identical in both games. The post-challenge ciphertext query is also handled identically in both games. The only difference in the two games is with respect to the key queries. In particular, for each key query x, the key components $\{\mathbf{t}_1^{(j^*,\beta)}\}_{\beta\in\{0,1\}}$ are computed differently in the two games. In Game 2, the challenger sets $\mathbf{t}_1^{(j^*,\beta)} = -\widetilde{\mathbf{y}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{s}_1^{(j^*,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j^*,\beta)} + \mathbf{e}_1^{(j^*,\beta)}$, while in Game 3.1.1, it sets $\mathbf{t}_1^{(j^*,\beta)} = -\widetilde{\mathbf{y}} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} + \mathbf{s}_1^{(j^*,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_1^{(j^*,\beta)} + \mathbf{e}_1^{(j^*,\beta)}$. Using the smudging lemma (Lemma 3.1.1), since $\sigma_{\mathsf{big}}/\sigma_{\mathsf{lwe}} \geq 2^\lambda$, we can argue that there exists a negligible function $\mathrm{negl}_{\mathsf{smud}}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $m \in \mathbb{N}$, $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) \leq 2m \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda)$, where

$$\mathcal{D}_1 \equiv \left\{ \left(\mathbf{e}_1^{(j^*,0)}, \mathbf{e}_1^{(j^*,1)} \right) : \ \mathbf{e}_1^{(j^*,\beta)} \leftarrow \chi_{\mathsf{big}}^m \text{ for } \beta \in \{0,1\} \right\},$$

$$\mathcal{D}_2 \equiv \left\{ \left(\widetilde{\mathbf{e}}_1^{(j^*,0)} + \mathbf{e}_1^{(j^*,0)}, \widetilde{\mathbf{e}}_1^{(j^*,0)} + \mathbf{e}_1^{(j^*,0)} \right) : \begin{array}{l} \mathbf{e}_1^{(j^*,\beta)} \leftarrow \chi_{\mathsf{big}}^m \text{ for } \beta \in \{0,1\}; \\ \widetilde{\mathbf{e}}_1^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m \text{ for } \beta \in \{0,1\} \end{array} \right\}.$$

As a result, if an adversary $\mathcal{A}$ makes $q_{\mathsf{keys}}(\lambda)$ key queries, then for any $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq q_{\mathsf{keys}}(\lambda) \cdot (2m \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda))$. ■

Lemma 7.4.4. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.1}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) \leq negl(\lambda)$.*

Proof. The main difference in these two games is in the key generation phase. In particular, for each key, the terms $\left(\mathbf{t}_{i^*+1}^{(j^*,0)}, \mathbf{t}_{i^*+1}^{(j^*,1)}\right)$ are computed differently in

both games. In Game $3.i^*.1$, $\mathbf{t}^{(j^*,\beta)}_{i^*+1} = -\mathbf{s}^{(j^*,\beta)}_{i^*} \cdot \mathbf{C}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \mathbf{s}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*+1}} + \mathbf{e}^{(j^*,\beta)}_{i^*+1}$, while in Game $3.i^*.2$, the challenger sets $\mathbf{t}^{(j^*,\beta)}_{i^*+1}$ as $-\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}^{(j^*,\beta)}_{\alpha} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}^{(\beta)}_{i^*+1}} + \mathbf{s}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*+1,\widetilde{x}_{i^*+1}} + \mathbf{e}^{(j^*,\beta)}_{i^*+1}$, which is equal to $-\mathbf{s}^{(j^*,\beta)}_{i^*} \cdot \mathbf{C}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \mathbf{s}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*+1}} + \mathbf{e}^{(j^*,\beta)}_{i^*+1} + \widetilde{\mathbf{e}}^{(j^*,\beta)}_{i^*+1} \cdot \mathbf{U}^{(\beta)}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{s}} \cdot \mathbf{E}$. In the second equality, $\widetilde{\mathbf{e}}^{(j^*,\beta)}_{i^*+1} \leftarrow \chi_{\mathsf{lwe}}$, $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$, $\mathbf{E}$ is sampled by Mixed-SubEnc from $\chi_{\mathsf{appr}}^{n\times m}$. The second equality follows by substituting the value of $\widetilde{\mathbf{t}}^{(j^*,\beta)}_{i^*} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}^{(j^*,\beta)}_{\alpha} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}^{(\beta)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}^{(\beta)}_{i^*}} + \mathbf{s}^{(j^*,\beta)}_{i^*} \cdot \mathbf{B}^{(j^*,\beta)}_{i^*,\widetilde{x}_{i^*}} + \widetilde{\mathbf{e}}^{(j^*,\beta)}_{i^*}$ (note that this is how $\widetilde{\mathbf{t}}^{(j^*,\beta)}_{i^*}$ is defined in Game $3.i^*.2$).

To prove this lemma, we will use the following fact, which follows from the smudging lemma (Lemma 3.1.1). Here, we use the fact that if $\mathbf{e}$ and $\mathbf{U}$ have entries bounded by $\sigma_{\mathsf{pre}} \cdot \mathsf{poly}(\lambda)$, then $\mathbf{e} \cdot \mathbf{U}$ can be "drowned" by a noise vector $\mathbf{e'}$ drawn from a noise distribution with parameter $\sigma_{\mathsf{pre}} \cdot 2^\lambda$.

Fact 7.4.1. *Let $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{lwe}}$ be families of distributions over $\mathbb{Z}$ as defined in the construction. For any polynomials $n(\cdot), m(\cdot)$, there exists a negligible function $\mathsf{negl}'(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $n = n(\lambda)$, $m = m(\lambda)$, $\chi_{\mathsf{big}} = \chi_{\mathsf{big}}(\lambda)$, $\chi_s = \chi_s(\lambda)$, $\chi_{\mathsf{appr}} = \chi_{\mathsf{appr}}(\lambda)$, $\chi_{\mathsf{lwe}} = \chi_{\mathsf{lwe}}(\lambda)$, and matrix $\mathbf{U} \in \mathbb{Z}_q^{m\times m}$ such that $\|\mathbf{U}\|_\infty \leq \sigma_{\mathsf{pre}} \cdot n$, $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) \leq \mathsf{negl}'(\lambda)$, where*

$$\mathcal{D}_1 = \left\{\mathbf{e} : \mathbf{e} \leftarrow \chi_{\mathsf{big}}^m\right\}; \quad \mathcal{D}_2 = \left\{ \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e_3} : \begin{array}{l} \mathbf{e}_1 \leftarrow \chi_{\mathsf{big}}^m; \mathbf{s} \leftarrow \chi_s^n; \mathbf{E} \leftarrow \chi_{\mathsf{appr}}^{n\times m}; \\ \mathbf{e}_2 = \mathbf{s} \cdot \mathbf{E}; \mathbf{e}'_3 \leftarrow \chi_{\mathsf{lwe}}^m; \\ \mathbf{e}_3 = \mathbf{e}'_3 \cdot \mathbf{U} \end{array} \right\}.$$

As a result, if an adversary $\mathcal{A}$ makes $q_{\mathsf{keys}}(\lambda)$ key queries, then for any $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.1}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) \leq 2q_{\mathsf{keys}}(\lambda) \cdot \mathrm{negl}'(\lambda)$. ■

Lemma 7.4.5. *Assuming the trapdoor generation algorithms* $\mathsf{LT}_{\mathsf{en}}$ *satisfy the* $(q, \sigma_{\mathsf{pre}})$*-row removal property, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) \leq negl(\lambda)$.

Proof. First, let us consider the differences between $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$.

1. Set $S^{(i^*)}$: In $\mathsf{Game}\ 3.i^*.2$, the challenger sets $S^{(i^*)} = [\lambda] \times \{0,1\}^2$, while in $\mathsf{Game}\ 3.i^*.3$, $S^{(i^*)} = ([\lambda] \setminus \{j^*\}) \times \{0,1\}^2$ ($\mathsf{tag}^*, \mathsf{tag}$ are chosen at the start of the security game, so j^* is well defined here). Also, $\widetilde{n}_{i^*} = \widetilde{n} = (4\lambda + w)n$ in $\mathsf{Game}\ 3.i^*.2$, while $\widetilde{n}_{i^*} = \widetilde{n} - 4n$ in $\mathsf{Game}\ 3.i^*.3$.

2. $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i=i^*}$ matrices: In $\mathsf{Game}\ 3.i^*.2$, the challenger chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q)$, while in $\mathsf{Game}\ 3.i^*.3$, the challenger chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}-4n}, 1^m, q)$. As a result, in $\mathsf{Game}\ 3.i^*.2$, it derives all matrices $\{\mathbf{B}_{i^*,b}^{(j,\beta)}\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2}$ from $\mathbf{M}_{i^*}$. In $\mathsf{Game}\ 3.i^*.3$, the challenger chooses $\{\mathbf{B}_{i^*,b}^{(j^*,\beta)}\}_{b,\beta\in\{0,1\}}$ uniformly at random, while the remaining are derived from $\mathbf{M}_{i^*}$.

3. Ciphertexts: Since the set $S^{(i^*)}$ is different in both games, the challenge and query ciphertexts are constructed differently in both games.

Let us now discuss why the row removal property is applicable here. In particular, we will focus on $\left(\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right)$. In $\mathsf{Game}\ 3.i^*.2$, each of these four matrices maps $\left[\mathbf{B}_{i^*,0}^{(j^*,0)} \mid \mathbf{B}_{i^*,1}^{(j^*,0)} \mid \mathbf{B}_{i^*,0}^{(j^*,1)} \mid \mathbf{B}_{i^*,1}^{(j^*,1)}\right]$ to a uniformly

random matrix. To see why, let us suppose $\mathsf{tag}^*_{j^*} = \beta^*$ and $\mathsf{tag}_{j^*} = 1 - \beta^*$. Then

- $\mathbf{B}^{(j^*,\beta^*)}_{i^*,0} \cdot \mathbf{U}^*_{i^*,0} = \mathbf{C}^{(j^*,\beta^*)}_{i^*,0}$, the rest are mapped to random matrices;
- $\mathbf{B}^{(j^*,\beta^*)}_{i^*,1} \cdot \mathbf{U}^*_{i^*,1} = \mathbf{C}^{(j^*,\beta^*)}_{i^*,1}$, the rest are mapped to random matrices;
- $\mathbf{B}^{(j^*,1-\beta^*)}_{i^*,0} \cdot \mathbf{U}_{i^*,0} = \mathbf{C}^{(j^*,1-\beta^*)}_{i^*,0}$, the rest are mapped to random matrices;
- $\mathbf{B}^{(j^*,1-\beta^*)}_{i^*,1} \cdot \mathbf{U}_{i^*,1} = \mathbf{C}^{(j^*,1-\beta^*)}_{i^*,1}$, the rest are mapped to random matrices.

Also, it is important to note that the $\{\mathbf{C}^{(j^*,\beta)}_{i^*,b}\}_{b,\beta\in\{0,1\}}$ are not used for responding to key generation queries. Therefore, we can use the row removal property to remove the rows corresponding to $\mathbf{B}^{(j^*,b)}_{i^*,\beta}$ from the level i^* matrices.

Suppose, on the contrary, that there exist an adversary $\mathcal{A}$ and a nonnegligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) \geq \eta(\lambda)$. We will use this adversary to build a reduction algorithm $\mathcal{B}$ that breaks the $(q, \sigma_{\mathsf{pre}})$-row removal property of $\mathsf{LT}_{\mathsf{en}}$.

The reduction algorithm first receives functionality index (k, w, L) from $\mathcal{A}$. Depending on the functionality index, the reduction algorithm sets $\ell = k \cdot L$, n, m, $\widetilde{n} = (4\lambda + w)n$ as in $\mathsf{Game}\ 3.i^*.2$ (and $\mathsf{Game}\ 3.i^*.3$) and sends these parameters to $\mathcal{A}$.

The reduction algorithm chooses $\mathsf{tag}^*, \mathsf{tag}$ and defines j^* as the first index where the two tags differ. For all $i \neq i^*$, $\mathcal{B}$ defines sets $S^{(i)}$ and samples matrices (with trapdoors) $\{\{\mathbf{B}^{(j,b)}_{i,\beta}\}_{(j,\beta,b)\in S^{(i)}}, \{\mathbf{P}_{i,v}\}_{v\in[w]}, T_i\}_{i\neq i^*}$ as in $\mathsf{Game}\ 3.i^*.2$

(and Game $3.i^*.3$). The reduction algorithm defines a set $S_{\mathcal{B}}$ which represents the set of rows that are removed in the transition between the two games. Formally, the reduction algorithm defines the sets

$$\mathsf{pos} = \{j : b, \beta \in \{0,1\}, (i^*, j^*, b, \beta) \text{ is at position } j \text{ in the set } \{i^*\}\times[\lambda]\times\{0,1\}^2\},$$

$$S_{\mathcal{B}} = \bigcup_{j \in \mathsf{pos}} \{n(j-1)+1, n(j-1)+2, \ldots, nj\}.$$

It sends $1^{\widetilde{n}}, 1^m, S_{\mathcal{B}}$ to the row removal property challenger. It receives $\mathbf{A}$ from the challenger, which it parses as

$$\mathbf{A} = \begin{bmatrix} \left\{\mathbf{B}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \left\{\mathbf{P}_{i^*,v}\right\}_{v\in[w]} \end{bmatrix}.$$

The reduction algorithm also chooses $(4\lambda+w-4)$ matrices $\{\mathbf{C}_{i,b}^{(j,\beta)}\}_{i\neq i^*, j\neq j^*}$ uniformly at random from $\mathbb{Z}_q^{n\times m}$.

Next, it receives the challenge programs $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$. It chooses $\gamma \leftarrow \{0,1\}$. For all $i \neq i^*$, it computes $(\mathbf{U}_{i,0}^*, \mathbf{U}_{i,1}^*)$ components by itself (this step is identical in both games). For $i = i^*$, it uses the row removal property challenger. It sets matrices $\mathbf{D}_b^{(j,\beta)}$ and $\widetilde{\mathbf{D}}_b^{(j,\beta)}$ as follows:

$$\forall\ (j,\beta,b) \in ([\lambda]\setminus\{j^*\})\times\{0,1\}^2, \quad \mathbf{D}_b^{(j,\beta)} = \begin{cases} \mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 0, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise,} \end{cases}$$

$$\widetilde{\mathbf{D}}_b^{(j,\beta)} = \begin{cases} \mathbf{C}_{i,b}^{(j,\beta)} & \text{if } \beta = \mathsf{tag}_j \text{ and } b = 1, \\ \left(\leftarrow \mathbb{Z}_q^{n\times m}\right) & \text{otherwise.} \end{cases}$$

Next, it sets matrices $\left\{\mathbf{Q}_{i,v}\right\}_{v\in[w]}$, $\{\widetilde{\mathbf{Q}}_{i,v}\}_{v\in[w]}$ as in Fig. 7.1 and sets matrices $\mathbf{W}$ and $\widetilde{\mathbf{W}}$ as

$$\mathbf{W} = \begin{bmatrix} \left\{\mathbf{D}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\mathbf{Q}_{i,v}\right\}_{v\in[w]} \end{bmatrix}, \quad \widetilde{\mathbf{W}} = \begin{bmatrix} \left\{\widetilde{\mathbf{D}}_b^{(j,\beta)}\right\}_{(j,\beta,b)\in S_\alpha} \\ \left\{\widetilde{\mathbf{Q}}_{i,v}\right\}_{v\in[w]} \end{bmatrix}.$$

It sends them as queries to the row removal challenger (note that $\mathbf{C}_{i^*,b}^{(j^*,\beta)}$ is not required for defining $\mathbf{W}$ and $\widetilde{\mathbf{W}}$). The challenger responds by sending $\mathbf{U}_{i^*,0}^*$ and $\mathbf{U}_{i^*,1}^*$, respectively. The reduction algorithm forwards $\left\{\left(\mathbf{U}_{i,0}^*, \mathbf{U}_{i,1}^*\right)\right\}_{i\in[\ell]}$ to the adversary. The ciphertext query is handled similarly, and the reduction algorithm receives $\{(\mathbf{U}_{i,0}, \mathbf{U}_{i,1})\}_{i\in[\ell]}$, which it forwards to $\mathcal{A}$ (the remaining ciphertext components can be computed by the reduction algorithm).

Next, the adversary sends polynomially many key queries. Note that the keys are generated in an identical manner in both games. Moreover, these keys can be generated without having $\{\mathbf{C}_{i^*,b}^{(j^*,\beta)}\}_{b,\beta}$ and the trapdoor for $\mathbf{M}_{i^*}$.

Finally, the adversary sends its guess, which the reduction algorithm forwards to the row removal property challenger. Clearly, if the row removal challenger chooses $b = 0$, then the reduction algorithm perfectly simulates $\mathsf{Game}\ 3.i^*.3$. If the challenger chooses $b = 1$, then the reduction algorithm perfectly simulates $\mathsf{Game}\ 3.i^*.2$ (here we use the fact that in $\mathsf{Game}\ 3.i^*.2$, each of the matrices $\left\{\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right\}$ maps $\left[\mathbf{B}_{i^*,0}^{(j^*,0)} \mid \mathbf{B}_{i^*,1}^{(j^*,0)} \mid \mathbf{B}_{i^*,0}^{(j^*,1)} \mid \mathbf{B}_{i^*,1}^{(j^*,1)}\right]$ to a uniformly random matrix).

Therefore, the reduction algorithm has advantage at least $\eta(\cdot)$. ■

Lemma 7.4.6. *Assuming the* $(n, q, \sigma_{\mathsf{lwe}})$-$\mathsf{LWE}$ *assumption holds, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $i^* \in [\ell]$*,* $\mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) \leq negl(\lambda)$*.*

Proof. In $\mathsf{Game}\ 3.i^*.3$, for each key query x, for each $\beta \in \{0,1\}$, the component $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot$

$\mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}$. In Game $3.i^*.4$, $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m$. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define q_{keys} hybrid experiments.

Hybrid $\boldsymbol{H_{o,0}}$ for $\boldsymbol{o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}}$ In this hybrid, for the first o keys, the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)}$ components are sampled uniformly at random in the first o queries, while the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,1)}$ components are sampled as in Game $3.i^*.3$. For the remaining $q_{\mathsf{keys}} - o$ key queries, the keys are generated as in Game $3.i^*.3$ in the remaining queries.

Hybrid $\boldsymbol{H_{o,1}}$ for $\boldsymbol{o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}}$ In this hybrid, for all keys, the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,0)}$ components are sampled uniformly at random. For the first o queries, the $\widetilde{\mathbf{t}}_{i^*}^{(j^*,1)}$ components are sampled uniformly at random, while the remaining are sampled as in Game $3.i^*.3$.

Clearly, $H_{0,0}$ corresponds to Game $3.i^*.3$, $H_{q_{\mathsf{keys}},1}$ is identical to Game $3.i^*.4$, and $H_{q_{\mathsf{keys}},0} \equiv H_{0,1}$. Let $a_{\mathcal{A},i,b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{i,b}$.

Claim 7.4.1. *Assuming the* $(n, q, \sigma_{\mathsf{lwe}})$-$\mathsf{LWE}$ *assumption, for any PPT adversary* $\mathcal{A}$ *making* $q_{\mathsf{keys}}(\cdot)$ *key queries, there exists a negligible function* $\mathsf{n}_{o,0}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ *and all indices* $o \in [q_{\mathsf{keys}}]$, $a_{\mathcal{A},o-1,0} - a_{\mathcal{A},o,0} \leq \mathsf{n}_{o,0}(\lambda)$.

Proof. Suppose there exist an adversary making q_{keys} key queries, and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, there exists an index $o \in [q_{\mathsf{keys}}]$

such that $a_{\mathcal{A},o-1,0} - a_{\mathcal{A},o,0} \geq \eta(\lambda)$. We will use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that breaks the $(n, q, \sigma_{\mathsf{lwe}})$-$\mathsf{LWE}$ assumption.

The reduction algorithm first receives $(1^k, 1^w, 1^L)$ from $\mathcal{A}$. It sets $\widetilde{n} = (4\lambda + w) \cdot n$. The reduction algorithm queries the LWE challenger m times and receives $\{(\mathbf{a}_i, u_i)\}_{i \leq m}$. It sets a matrix $\mathbf{A} = [\mathbf{a}_1^T \ \mathbf{a}_2^T \dots \mathbf{a}_m^T]$ (that is, $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$) and $\mathbf{u} = [u_1 u_2 \dots u_m]$ (that is, $\mathbf{u} \in \mathbb{Z}_q^m$).

The reduction algorithm then chooses two tags, $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$, and let j^* be the first index where they differ. Next, the reduction algorithm defines set $S^{(i)}$ for each i, set $\hat{S}$, matrices $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{(i,j,\beta,b) \in \hat{S}}$, $\{\mathbf{P}_{i,v}\}_{i \in [\ell], v \in [w]}$ and $\{T_i\}_{i \in [\ell]}$ as in Game $3.i^*.3$ (and Game $3.i^*.4$). Note that $(i^*, j^*, \beta, b) \notin \hat{S}$ for $b, \beta \in \{0,1\}$. The reduction algorithm chooses $\mathbf{B}_{i^*,b}^{(j^*,1)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $b \in \{0,1\}$. It sends the public parameters to $\mathcal{A}$.

The adversary sends two challenge functions, $\mathsf{BP}^{(0)}, \mathsf{BP}^{(1)}$, and a ciphertext query BP. Note that in Game $3.i^*.3$ (and Game $3.i^*.4$), the challenge and query ciphertext are computed identically, and the reduction algorithm has all the matrices/trapdoors required for computing the ciphertext components.

Next, after receiving the challenge ciphertext and the query ciphertext, the adversary queries for secret keys. For the first $o-1$ secret keys, the reduction algorithm responds as in $H_{o-1,0}$ (which is identical to the response in $H_{o,0}$). In particular, to handle these key queries, the reduction algorithm does not require $\mathbf{B}_{i^*,\beta}^{(j^*,b)}$, since in both hybrids $\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m$. For the oth query, the reduction algorithm receives $\mathbf{x} \in \{0,1\}^k$. It sets $\widetilde{\mathbf{x}}$ by repeating

the input L times, and sets $\mathbf{B}^{(j^*,0)}_{i^*,\widetilde{x}_{i^*}} = \mathbf{A}$ (the LWE public matrix) and chooses $\mathbf{B}^{(j^*,0)}_{i^*,1-\widetilde{x}_{i^*}} \leftarrow \mathbb{Z}_q^{n\times m}$ (this might be used for the later key queries). The reduction algorithm sets

$$\widetilde{\mathbf{t}}^{(j^*,0)}_{i^*} = -\sum_{\alpha=1}^{i^*-1}\left(\widetilde{\mathbf{t}}^{(j^*,0)}_{\alpha}\cdot\prod_{\delta=\alpha}^{i^*-1}\mathbf{U}^{(0)}_{\delta,\widetilde{x}_\delta}\right) - \widetilde{\mathbf{y}}\cdot\prod_{\delta=1}^{i^*-1}\mathbf{U}^{(0)}_{\delta,\widetilde{x}_\delta} + \widetilde{\mathbf{s}}\cdot\mathbf{P}_{i^*,\mathsf{st}^{(0)}_{i^*}} + \mathbf{u}.$$

It computes $\widetilde{\mathbf{t}}^{(j^*,1)}_{i^*}$ as in $H_{o-1,0}$ (and $H_{o,0}$). All the remaining key queries are handled identically in $H_{o-1,0}$ and $H_{o,0}$, and the reduction algorithm has all the matrices required to compute them (in particular, after responding to the oth query, all $\{\mathbf{B}^{(j^*,\beta)}_{i^*,b}\}_{(b,\beta)\in\{0,1\}^2}$ are well defined).

Finally, the adversary sends its guess, and the reduction algorithm forwards it to the LWE challenger. Clearly, if $\mathbf{u}$ is a uniformly random vector, then so is the $\widetilde{\mathbf{t}}^{(j^*,0)}_{i^*}$ component for the oth query, and therefore $\mathcal{B}$ perfectly simulates $H_{o,0}$. If $\widetilde{\mathbf{t}}^{(j^*,0)}_{i^*} = \mathbf{s}\cdot\mathbf{B}^{(j^*,0)}_{i^*,\widetilde{x}_i} + \widetilde{\mathbf{e}}^{(j^*,0)}_{i^*}$, then the reduction algorithm implicitly sets $\mathbf{s}^{(j^*,0)}_{i^*} = \mathbf{s}$. Also, note that $\mathbf{s}^{(j^*,0)}_{i^*}$ is chosen afresh for each key query, and hence $\mathbf{s}$ will not be required anywhere else in simulating $H_{o-1,0}$. Therefore the reduction algorithm perfectly simulates $H_{o-1,0}$. As a result, it breaks the LWE assumption with advantage η.

■

Claim 7.4.2. *Assuming the* $(n, q, \sigma_{\mathsf{lwe}})$-$\mathsf{LWE}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *making* $q_{\mathsf{keys}}(\cdot)$ *key queries, there exists a negligible function* $\mathsf{n}_{o,1}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ *and all indices* $o \in [q_{\mathsf{keys}}]$, $a_{\mathcal{A},o-1,1} - a_{\mathcal{A},o,1} \leq \mathsf{n}_{o,1}(\lambda)$.

This proof is identical to the proof of Claim 7.4.1.

Using the above claims, it follows that for any PPT adversary, there exists a negligible function $\mathrm{negl}_{3.i^*.4}$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) \leq \mathrm{negl}_{3.i^*.4}(\lambda)$. ■

Lemma 7.4.7. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $negl_{3.(i^*+1).1}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.(i^*+1).1}(\lambda) \leq negl_{3.(i^*+1).1}(\lambda)$.*

Proof. The only difference between Game $3.i^*.4$ and Game $3.(i^*+1).1$ is that for each key query, the components $\left(\mathbf{t}_{i^*+1}^{(j^*,0)}, \mathbf{t}_{i^*+1}^{(j^*,1)}\right)$ are computed differently. In particular, in Game $3.(i^*+1).1$, the term $\widetilde{\mathbf{t}}_{i^*+1}^{(j^*,\beta)}$ has an additional term $\widetilde{\mathbf{e}}_{i^*+1}^{(j^*,\beta)}$ which is drawn from the χ_{lwe}^m distribution.

The proof of this lemma is identical to the proof of Lemma 7.4.3 by setting $\mathrm{negl}_{3.(i^*+1).1}(\cdot) = q_{\mathsf{keys}}(\cdot) \cdot (2m \cdot \mathrm{negl}_{\mathsf{smud}}(\cdot))$ (recall that $\mathrm{negl}_{\mathsf{smud}}(\cdot)$ is the negligible function given by Lemma 3.1.1). ■

Next, we will look at Game $4.i$ for $i \in [\ell]$. For notational convenience, we refer to Game 4 as Game 4.0. First, recall that Game 4.0 is identical to Game $3.\ell.4$. Note that in Game 4.0, the challenger uses $\{\mathbf{P}_{1,v}\}_{v\in[w]}$ only for computing $\left(\mathbf{U}_{1,0}^*, \mathbf{U}_{1,1}^*\right)$ (for the challenge ciphertext) and $(\mathbf{U}_{1,0}, \mathbf{U}_{1,1})$ (for the ciphertext query). In particular, it is not used in the key generation phase. More generally, for all $i \in [\ell]$, $\{\mathbf{P}_{i,v}\}_{v\in[w]}$ is used in Game $4.(i-1)$ only for computing $\left(\mathbf{U}_{i,0}^*, \mathbf{U}_{i,1}^*\right)$ and $(\mathbf{U}_{i,0}, \mathbf{U}_{i,1})$. This observation is useful for the following lemma.

Lemma 7.4.8. *Assuming* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* $(q, \chi_{\mathsf{appr}}, \sigma_{\mathsf{pre}})$*-target switching property, then for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl_{4.i^*}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},4.(i^*-1)} - \mathsf{Adv}_{\mathcal{A},4.i^*} \leq negl_{4.i^*}(\lambda)$.

Proof. The only difference between Game $4.(i^*-1)$ and Game $4.i^*$ is with respect to the matrices $\left(\mathbf{U}^*_{i^*,0}, \mathbf{U}^*_{i^*,1}\right)$ (in the challenge ciphertext) and $(\mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1})$ (in the ciphertext query). In Game $4.(i^*-1)$, these matrices are computed using Mixed-SubEnc, while they are computed using Mixed-SubEnc* in Game $4.i^*$. Recall that the only difference between the Mixed-SubEnc and Mixed-SubEnc* ciphertext components is that the Mixed-SubEnc* outputs map the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ matrices to uniformly random matrices (instead of mapping to $\left\{\mathbf{P}_{i^*+1,\pi(v)}\right\}_{v\in[w]}$ as in Mixed-SubEnc). An important point to note is that the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ matrices are not used anywhere else in both games. In particular, note that $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ are not used for computing $\left\{\mathbf{U}^*_{i^*-1,b}, \mathbf{U}_{i^*-1,b}\right\}_{b\in\{0,1\}}$.

Suppose there exist an adversary $\mathcal{A}$ and a nonnegligible function $\eta(\cdot)$ such that $\mathsf{Adv}_{4.i^*-1}(\lambda) - \mathsf{Adv}_{4.i^*}(\lambda) \geq \eta(\lambda)$. We will construct a reduction algorithm that breaks the target switching property with advantage $\eta(\cdot)$.

Setup phase. The reduction algorithm first performs the following steps from the setup phase, which are common for both Game $4.(i^*-1)$ and Game $4.i^*$. It defines $\widetilde{n}_i$, $S^{(i)}$ for all $i \in [\ell]$, chooses $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$, and defines j^* as the first index where tag^* and tag differ. Next, it defines $\left\{\{\mathbf{B}^{(j,\beta)}_{i,b}\}_{(j,\beta,b)\in S^{(i)}}, \{\mathbf{P}_{i,v}\}_{v\in[w]}, T_i\right\}_{i\neq i^*}$ as in Game $4.(i^*-1)$. It also defines $\hat{S}$

and chooses $\{\mathbf{C}_{i,b}^{(j,\beta)}\}_{(i,j,\beta,b)\in\hat{S}}$ as in Game $4.(i^*-1)$.

The reduction algorithm sets $\widetilde{k} = \widetilde{n}_i - w \cdot n$ and queries the target switching property challenger by sending $1^{\widetilde{n}_i}, 1^m$ and setting $S_{\mathcal{B}} = [\widetilde{k}]$. It receives a matrix $\mathbf{A} \in \mathbb{Z}_q^{\widetilde{k}\times m}$ and parses $\mathbf{A}$ as follows:

$$\mathbf{A} = \left[\left\{\mathbf{B}_{j,b}^{(1,\beta)}\right\}_{(j,\beta,b)\in S^{(i^*)}}\right].$$

Challenge phase. The reduction algorithm receives $\mathsf{BP}^{(1)}$, $\mathsf{BP}^{(2)}$. It chooses $\gamma \leftarrow \{0,1\}$, and for all $\alpha < i^*$ it computes $\left(\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right)$ using Mixed-SubEnc* (as in Game $4.(i^*-1)$ and Game $4.i^*$). Note in particular that $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ are not used for computing these matrices. It then sends its target switching property query matrices $\mathbf{Z}_{0,b}^*, \mathbf{Z}_{1,b}^*$ of dimensions $\widetilde{n}_i \times m$ defined below.

$$\mathbf{Z}_{0,b}^* = \begin{bmatrix} \left\{\mathbf{C}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i^*)}} \\ \left\{\mathbf{P}_{i^*,\pi_{i^*,b}^{\gamma}(v)}\right\}_{v\in[w]} \end{bmatrix} \mathbf{Z}_{1,b}^* = \begin{bmatrix} \left\{\mathbf{C}_{i^*,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i^*)}} \\ \leftarrow \mathbb{Z}_q^{w\cdot n\times m} \end{bmatrix}.$$

It receives $\mathbf{U}_{i^*,b}^*$ from the challenger.

Query phase. The ciphertext query is handled similarly to the challenge ciphertext. The key queries are handled identically in both Game $4.(i^*-1)$ and Game $4.i^*$. ■

Lemma 7.4.9. *For any adversary $\mathcal{A}$,* $\mathsf{Adv}_{\mathcal{A},5} = 0$.

Proof. We will argue that any adversary $\mathcal{A}$ has advantage 0 in Game 5. First, note that the challenge phase uses Mixed-SubEnc*. As a result, it does not have

any information about the choice $\gamma \leftarrow \{0,1\}$. Next, in the key query phase, for each key query, the challenger chooses $\mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}}$, which might depend on γ. However, the important point here is that for each key query x, both challenge programs have identical output. Therefore, the $\mathbf{P}^{(\beta)}_{\ell+1,\mathsf{st}^{(\beta)}_{\ell+1}}$ matrices are independent of γ. As a result, the adversary has zero advantage in Game 5.

■

7.4.3 1-query restricted accept indistinguishability

In order to prove restricted accept indistinguishability security, we take a slightly different approach. First, we show that our construction achieves *complete* accept indistinguishability security, which is defined as follows.

Definition 7.4.1 (q-query complete accept indistinguishability)**.** Let $q(\cdot)$ be any fixed polynomial. A mixed FE scheme Mixed-FE = (Setup, Enc, SK-Enc, KeyGen, Dec) is said to satisfy q-query *complete* accept indistinguishability security if there exist algorithms $\mathsf{SK\text{-}Enc}^*, \mathsf{KeyGen}^*$ such that for every stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{O_1^b(\cdot),O_2^b(\cdot)}(\mathsf{pp},\mathsf{ct}_b)=b : \begin{array}{c}(1^\kappa, f^*) \leftarrow \mathcal{A}(1^\lambda);\\ (\mathsf{pp},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda,1^\kappa);\\ b \leftarrow \{0,1\};\ \mathsf{ct}_1 \leftarrow \mathsf{SK\text{-}Enc}(\mathsf{msk}, f^*);\\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pp})\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where

- oracle $O_1^0(\cdot) = \mathsf{KeyGen}^*(\mathsf{pp},\cdot)$, $O_1^1(\cdot) = \mathsf{KeyGen}(\mathsf{msk},\cdot)$,
  $O_2^0(\cdot) = \mathsf{SK\text{-}Enc}^*(\mathsf{pp},\cdot)$, $O_2^1(\cdot) = \mathsf{SK\text{-}Enc}(\mathsf{msk},\cdot)$;

- $\mathcal{A}$ can make at most $q(\lambda)$ queries to the $O_2^b(\cdot)$ oracle;
- every secret key query m made by adversary $\mathcal{A}$ to the $O_1^b(\cdot)$ oracle must satisfy the condition that $f^*(m) = 1$ as well as $f(m) = 1$ for every ciphertext query f made by $\mathcal{A}$ to the $O_2^b(\cdot)$ oracle; and
- $\mathcal{A}$ must make all (at most $q(\lambda)$) $O_2^b(\cdot)$ oracle queries before making any query to the $O_1^b(\cdot)$ oracle.

At a high level, this states that if the adversary only queries for keys for inputs m and ciphertexts for functions f such that $f(m) = 1$ on all combinations, then there exist special encryption and key generation algorithms $(\mathsf{SK\text{-}Enc}^*, \mathsf{KeyGen}^*)$ such that they only take public parameters as inputs, and the adversary cannot distinguish between correctly computed keys and ciphertexts from these (simulated) special keys and ciphertexts.

Below we provide a sequence of hybrid games that we later use to argue complete accept indistinguishability security. To complete the argument, later (in Section 7.4.5) we simply argue that complete accept indistinguishability implies restricted accept indistinguishability.

Game 0 This corresponds to the original 1-query restricted accept indistinguishability security game in which the challenger encrypts the challenge branching program BP^* sent by the adversary.

- **Setup phase.** The adversary sends the functionality index (k, w, L)

and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$.
2. Next, it samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as
$$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$
3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.
4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger chooses a random λ-bit string $\mathsf{tag}^* \leftarrow \{0,1\}^\lambda$. Let
$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$
The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,
$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array} \right).$$
Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger chooses a λ-bit string $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and responds as follows:

     (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\} \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

     $$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

     (b) It then chooses secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$, error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ as

     $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\},$$
$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 1 This is identical to the previous game, except the challenger now chooses both tags tag^* and tag at the beginning during the setup phase, and it aborts if $\mathsf{tag}^* = \mathsf{tag}$.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.

2. Next, it samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

   $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda - 1]$. Next, it sets vector $\mathbf{y}^{(\lambda)}$ as

   $$\mathbf{y}^{(\lambda)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j\in[\lambda-1]} \mathbf{y}^{(j)}.$$

   (b) It then chooses secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$, error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ as

   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
   $$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
   $$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\},$$

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Notation In all the following hybrid games, let diff denote the set of indices j such that $\mathsf{tag}_j^* \neq \mathsf{tag}_j$. Similarly, let comm denote the set of indices j such that $\mathsf{tag}_j^* = \mathsf{tag}_j$. Concretely, in all the following hybrids, sets diff and comm are defined as follows:

$$\mathsf{diff} \stackrel{\text{def}}{=} \left\{j \in [\lambda] \ :\ \mathsf{tag}_j^* \neq \mathsf{tag}_j\right\}, \quad \mathsf{comm} \stackrel{\text{def}}{=} \mathsf{comm}.$$

Additionally, let j^* denote the smallest index in diff (i.e., $j^* = \min_{j\in\mathsf{diff}} j$), and let $\beta^* = \mathsf{tag}_{j^*}^*$. Note that since the challenger aborts whenever $\mathsf{tag}^* = \mathsf{tag}$, thus j^*, β^* always exist whenever the challenger does not abort. Also, we will use $\widehat{\mathsf{diff}}$ to denote the set diff excluding index j^*, i.e., $\widehat{\mathsf{diff}} = \mathsf{diff} \setminus \{j^*\}$.

Game 2 This is identical to the previous game, except the challenger, while answering a secret key query, now puts the $\widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1}$ component in $\mathbf{y}^{(j^*)}$ instead of $\mathbf{y}^{(\lambda)}$, and the rest are sampled uniformly at random.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below.

  2. Next, it samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\, i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $i \in [\ell], j \in [\lambda], \beta, b \in \{0,1\}$.

  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

  $$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w] \right),$$
  $$S^* = [\ell] \times [\lambda] \times \{0,1\}^2.$$

  The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1)

as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = [\ell] \times [\lambda] \times \{0,1\}^2$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$\left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors

as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$. Next, it sets vector $\mathbf{y}^{(j^*)}$ as

$$\mathbf{y}^{(j^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}_{1,1} - \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)}.$$

(b) It then chooses secret vectors $\{\mathbf{s}_i^{(j,\beta)}\}_{i,j,\beta}$, error vectors $\{\mathbf{e}_i^{(j,\beta)}\}_{i,j,\beta}$ as

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m.$$

(c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\},$$
$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of 4ℓ hybrid experiments, $\mathsf{Game}\ 3.i^*.\{1,2,3,4\}$ for $i^* = 1$ to ℓ.

$\underline{\mathsf{Game}\ 3.i^*.1}$ In hybrids $\mathsf{Game}\ 3.i^*.1$, the $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices for all diff strands and levels $i < i^*$ are *not* sampled (at all) along with other level i matrices; ciphertext components for levels $i < i^*$ are used to target only the remaining

matrices; i.e., the ciphertext matrices do not target $\mathbf{B}_{i,b}^{(j,\beta)}$ matrices for $j \in \mathsf{diff}$ and $i < i^*$ to some prespecified $\mathbf{C}_{i,b}^{(j,\beta)}$ or random matrices. Also, the first $i^* - 1$ components in each secret key are set to be uniformly random vectors, and the next component is hardwired such that correctness holds, and some smudgeable noise is also introduced in these components. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i < i^*, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2,$$
$$\forall\ i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

  Also, let

$$\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^*, \\ \widetilde{n} & \text{for } i \geq i^*. \end{cases}$$

  Set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

(a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [i^*-1]\times\mathsf{diff}\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (j,\beta) \in \mathsf{diff}\times\{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP* after i^*-1 steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1]\times\mathsf{comm}\times\{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^{*} & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$

$$\begin{aligned} &\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \\ &\mathbf{t}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &\qquad + \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} + \mathbf{e}_{i^*}^{(j,\beta)}, \\ &\forall\ \beta \in \{0,1\}, \\ &\mathbf{t}_{i^*}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &\qquad + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}, \end{aligned}$$

$$\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*]) \times \mathsf{diff} \times \{0,1\},$$

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game $3.i^*.2$ This is identical to the previous game, except the (i^*+1)th key component in all diff strands is also hardwired. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i < i^*, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2,$$
  $$\forall\ i \geq i^*, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

  Also, let

  $$\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^*, \\ \widetilde{n} & \text{for } i \geq i^*. \end{cases}$$

  Set $\hat{S} = \big\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\big\}$.

  2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi^*_{i,b} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \\ \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^*\end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \\ \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,$$
$$\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,$$
$$\forall\ (j,\beta) \in \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after i^*-1 steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j\in[\lambda]\setminus\{j^*\}} \mathbf{y}^{(j)},$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

Also, for $(j,\beta) \in \mathsf{diff} \times \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j,\beta)} = \mathbf{0}^{n\times m}$, and let $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)}$ denote the following vector. For all $j \in \widehat{\mathsf{diff}}$,

$$\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}$$

$$
\begin{aligned}
&+\mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)}, \\
\widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = &- \sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\
&+\widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}.
\end{aligned}
$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$
\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1, \end{cases}
$$

$$
\forall\ (i,j,\beta) \in [i^* - 1] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},
$$

$$
\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*}^{(j,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} + \mathbf{e}_{i^*}^{(j,\beta)},
$$

$$
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)},
$$

$$
\begin{aligned}
\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j,\beta)} = &- \sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) \\
&+\mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\
&+ \mathbf{s}_{i^*+1}^{(j,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\beta)} + \mathbf{e}_{i^*+1}^{(j,\beta)},
\end{aligned}
$$

$$
\begin{aligned}
\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} = &- \sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\
&+ \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)},
\end{aligned}
$$

$$\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \mathsf{diff} \times \{0,1\},$$

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game $3.i^*.3$ This is identical to the previous game, except $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for diff strands and levels $i = i^*$ are *not* sampled along with other level i^* matrices, but instead they are sampled uniformly at random. Also, ciphertext components for level i^* are used to target only the remaining matrices. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i < i^*+1, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2,$$

$$\forall\ i \geq i^* + 1, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

Also, let

$$\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^* + 1, \\ \widetilde{n} & \text{for } i \geq i^* + 1. \end{cases}$$

Set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$

$$\forall\ (j,\beta,b) \in \mathsf{diff} \times \{0,1\}^2, \quad \mathbf{B}_{i^*,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$

$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

     $$\begin{aligned}
     &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
     &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
     &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
     &\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,
     \end{aligned}$$

$$\forall\ (j,\beta) \in \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after i^*-1 steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

Also, for $(j,\beta) \in \mathsf{diff} \times \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j,\beta)} = \mathbf{0}^{n \times m}$, and let $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)}$ denote the following vector. For all $j \in \widehat{\mathsf{diff}}$,

$$\begin{aligned} \widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} = &- \sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &+ \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)}, \\ \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} = &- \sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &+ \ \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j^*,\beta)}. \end{aligned}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j,\beta) \in [i^*-1] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*}^{(j,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} + \mathbf{e}_{i^*}^{(j,\beta)},$$
$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{i^*}^{(j^*,\beta)} + \mathbf{e}_{i^*}^{(j^*,\beta)}.$$

For all $(j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}$,

$$\begin{aligned}\mathbf{t}_{i^*+1}^{(j,\beta)} =& -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &+ \mathbf{s}_{i^*+1}^{(j,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\beta)} + \mathbf{e}_{i^*+1}^{(j,\beta)}.\end{aligned}$$

For all $\beta \in \{0,1\}$,

$$\begin{aligned}\mathbf{t}_{i^*+1}^{(j^*,\beta)} =& -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \\ &+ \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathbf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)},\end{aligned}$$

$\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \mathsf{diff} \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ'.

Game $3.i^*.4$ This is identical to the previous game, except the i^*th level key component in diff strands is a uniformly random n length vector, i.e., all first i^* level components in diff strands are random elements. Also, we no longer sample the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for diff strands and levels $i = i^*$ at all. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i < i^* + 1, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2,$$
  $$\forall\ i \geq i^* + 1, \quad S^{(i)} = [\lambda] \times \{0,1\}^2.$$

  Also, let

  $$\widetilde{n}_i = \begin{cases} \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i < i^* + 1, \\ \widetilde{n} & \text{for } i \geq i^* + 1. \end{cases}$$

  Set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.

  2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i, j, \beta, b) \in \hat{S}$.

  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi^*_{i,b} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [i^*] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (j,\beta) \in \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
\end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{i^*}^{(1-\beta^*)}, \mathsf{st}_{i^*}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after $i^* - 1$ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell + 1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

   Also, for $(j,\beta) \in \mathsf{diff} \times \{0,1\}$, let $\mathbf{B}_{\ell+1,\widetilde{x}_{\ell+1}}^{(j,\beta)} = \mathbf{0}^{n \times m}$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples

$(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1, \end{cases}$$

$$\forall\ (i,j,\beta) \in [i^*] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{s}_{i^*+1}^{(j,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j,\beta)} + \mathbf{e}_{i^*+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{i^*+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{i^*} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*+1,\mathsf{st}_{i^*+1}^{(\beta)}} + \mathbf{s}_{i^*+1}^{(j^*,\beta)} \cdot \mathbf{B}_{i^*+1,\widetilde{x}_{i^*+1}}^{(j^*,\beta)} + \mathbf{e}_{i^*+1}^{(j^*,\beta)},$$

$\forall\ (i,j,\beta) \in ([\ell+1] \setminus [i^*+1]) \times \mathsf{diff} \times \{0,1\}$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell + 1. \end{cases}$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 4 This is similar to Game $3.\ell.4$, except that the terms $\mathbf{t}_{\ell+1}^{(j^*,\beta)}$ have an additional small error $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)}$, which is smudged by the main error term $\mathbf{e}_{\ell+1}^{(j^*,\beta)}$.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

  $$\forall\ i \in [\ell], \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2.$$

  Also, let $\widetilde{n}_i = \widetilde{n} - |\mathsf{diff}| \cdot 4n$ for all $i \in [\ell]$.

  Set $\hat{S} = \left\{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j, \beta, b) \in S^{(i)}\right\}$.

  2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

  $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i, j, \beta, b) \in \hat{S}$.

  4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

  $$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$

$S^* = \hat{S}$.

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$\left(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$

$$\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$

$$\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,$$

$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps, respectively. Also, let $\Gamma, \widetilde{\mathbf{y}}$, and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{y}} = \sum_{j \in [\lambda] \setminus \{j^*\}} \mathbf{y}^{(j)},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing the query ciphertext. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \leq \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j^*,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta)}}^{(\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 5 This is identical to the previous game, except the $(\ell+1)$th key components in the special strand (i.e., j^*th strand) are random elements. We describe this in detail below.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction.

Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i \in [\ell], \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2.$$

Also, let $\widetilde{n}_i = \widetilde{n} - |\mathsf{diff}| \cdot 4n$ for all $i \in [\ell]$.

Set $\hat{S} = \left\{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\right\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell], b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S,\\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S},\\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]},\\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

     $$\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n,$$
     $$\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,$$
     $$\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,$$

$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps, respectively. Also, let Γ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing the query ciphertext. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all tuples $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}.$$

For all $(j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}$,

$$\mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of ℓ hybrid experiments, $\mathsf{Game}\ 5.i^*$ for $i^* = 1$ to ℓ.

$\underline{\mathsf{Game}\ 5.i^*}$ This is identical to the previous game, except matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ for $j \in \mathsf{comm}, \beta = 1 - \mathsf{tag}_j$ strands (i.e., strands in which $\mathbf{B}_{i,b}^{(j,\beta)}$ were targeting random matrices themselves) and levels $i \le i^*$ are *not* sampled along with other level $i \le i^*$ matrices, but instead they are sampled uniformly at random. Also, ciphertext components for levels $i \le i^*$ are used to target only the remaining matrices. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

$$\forall\ i \le i^*, \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\},$$

$$\forall\ i > i^*, \quad S^{(i)} = \mathsf{comm} \times \{0,1\}^2.$$

Also, let

$$\widetilde{n}_i = \begin{cases} (2 \cdot |\mathsf{comm}| + w)n & \text{for } i \le i^*, \\ \widetilde{n} - |\mathsf{diff}| \cdot 4n & \text{for } i > i^*. \end{cases}$$

Set $\hat{S} = \{(i,j,\beta,b) \in [\ell] \times [\lambda] \times \{0,1\}^2 \ : \ (j,\beta,b) \in S^{(i)}\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$

$$\forall\ (i,j,\beta,b) \in ([i^*] \times \mathsf{comm} \times \{0,1\}^2) \setminus \hat{S}, \quad \mathbf{B}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i \in [\ell], b \in \{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w] \right),$$

$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc} \left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b) \in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v) \in [\ell] \times [w]}, \\ \{T_i\}_{i \in [\ell]}, \mathsf{BP}^* \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \{\mathbf{U}_{i,b}^*\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

  1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

     (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

     $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

     (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

  2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

     (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

     (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

     $$\begin{aligned} &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\ &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\ &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\ &\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \end{aligned}$$

$$\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps, respectively. Also, let Γ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = [\ell+1] \times \mathsf{comm} \times \{0,1\},$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing the query ciphertext. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}.$$

For all $(j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}$,

$$\mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of ℓ hybrid experiments, $\mathsf{Game}\ 6.i^*$ for $i^* = 2$ to $\ell + 1$.

$\underline{\mathsf{Game}\ 6.i^*}$ This is identical to the previous game (i.e., $\mathsf{Game}\ 5.\ell$), except in the comm strands, for which we sample $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices uniformly at random, the key components for levels $i \leq i^*$ are random elements. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

     $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j, \beta, b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

     Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2\ :\ (j, \beta, b) \in S^{(i)}\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q),$$
$$\forall\ (i,j,\beta,b) \in ([\ell] \times \mathsf{comm} \times \{0,1\}^2) \setminus \hat{S}, \quad \mathbf{B}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}.$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

   $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

   (b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

   $$\begin{aligned}
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
   &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
   &\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
   &\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} \leftarrow \mathbb{Z}_q^m, \\
   &\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
   \end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP^* after ℓ steps, respectively. Also, let Γ and $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = ([\ell+1] \setminus [i^*]) \times \mathsf{comm} \times \{0,1\} \cup \left\{(i^*, j, \beta) : j \in \mathsf{comm}, \beta = \mathsf{tag}_j\right\},$$

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing the query ciphertext. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows. For all $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} \mathbf{s}_1^{(j,\beta)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)} & \text{if } i = 1, \\ -\mathbf{s}_{i-1}^{(j,\beta)} \cdot \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \cdot \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \mathbf{t}_i^{(j,1-\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} + \mathbf{e}_i^{(j,1-\mathsf{tag}_j)},$$

$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}.$$

For all $(j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}$,

$$\mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of $\ell + 1$ hybrid experiments, $\mathsf{Game}\ 7.i^*$ for $i^* = 1$ to $\ell + 1$.

$\underline{\mathsf{Game}\ 7.i^*}$ This is identical to the previous game (i.e., $\mathsf{Game}\ 6.(\ell+1)$), except in the comm strands, for which we *still* sample $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ matrices using $\mathsf{EnTrapGen}$, the key components for levels $i \leq i^*$ are random elements. Also, we no longer sample the matrices $\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}$ at all, which were sampled uniformly at random in the previous game. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

     $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j, \beta, b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

     Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2\ :\ (j, \beta, b) \in S^{(i)}\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

(a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

(a) It chooses a secret vector as $\widetilde{\mathbf{s}} \leftarrow \chi_s^n$ and $\lambda - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in [\lambda] \setminus \{j^*\}$.

(b) It then chooses vectors $\mathbf{s}_i^{(j,\beta)}, \mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{s}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^n, \\
&\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
&\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
&\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (i,j) \in [\ell+1] \times \mathsf{comm}, \quad \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \widetilde{\mathbf{t}}_i^{(j,\mathsf{tag}_j)} \leftarrow \mathbb{Z}_q^m, \\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathsf{st}_{\ell+1}^{(1-\beta^*)}, \mathsf{st}_{\ell+1}^{(\beta^*)}$ denote the state of branching programs BP, BP* after ℓ steps, respectively. Also, let Γ and

$\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\Gamma = \left\{(i,j,\beta) \in ([\ell+1] \setminus [i^*]) \times \mathsf{comm} \times \{0,1\} \ : \ \beta = \mathsf{tag}_j\right\},$$
$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

For $v \in [w]$, let $\mathbf{P}_{\ell+1,v}^{(\beta^*)}$ be the top level matrices chosen while computing the challenge ciphertext. Similarly, let $\mathbf{P}_{\ell+1,v}^{(1-\beta^*)}$ be the top level matrices chosen while computing the query ciphertext. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j) \in [i^*] \times \mathsf{comm}, \quad \mathbf{t}_i^{(j,\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_i^{(j,\mathsf{tag}_j)} + \mathbf{e}_i^{(j,\mathsf{tag}_j)}.$$

For all $(i,j,\beta) \in \Gamma$,

$$\mathbf{t}_i^{(j,\beta)} = \begin{cases} -\mathbf{s}_{i-1}^{(j,\beta)} \mathbf{C}_{i-1,\widetilde{x}_{i-1}}^{(j,\beta)} + \mathbf{s}_i^{(j,\beta)} \mathbf{B}_{i,\widetilde{x}_i}^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)} & \text{if } 1 < i \le \ell, \\ -\mathbf{s}_\ell^{(j,\beta)} \cdot \mathbf{C}_{\ell,\widetilde{x}_\ell}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)} & \text{if } i = \ell+1, \end{cases}$$

$$\forall\ (i,j) \in [\ell+1] \times \mathsf{comm}, \quad \mathbf{t}_i^{(j,1-\mathsf{tag}_j)} = \widetilde{\mathbf{t}}_i^{(j,1-\mathsf{tag}_j)} + \mathbf{e}_i^{(j,1-\mathsf{tag}_j)},$$
$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}.$$

For all $(j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}$,

$$\mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta) \in [\ell+1] \times [\lambda] \times \{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 8 This is identical to the previous game (i.e., Game $7.(\ell+1)$). For ease of exposition, we describe it in detail below.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

     $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j, \beta, b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

     Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2\ :\ (j, \beta, b) \in S^{(i)}\}$.

  2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

     $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

  3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i, j, \beta, b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi^*_{i,b} : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}^*_{\alpha,0}, \mathbf{U}^*_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S^*}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \\ \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}^*\end{array}\right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}^*_{i,b}\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = \hat{S}$. It runs the Mixed-SubEnc routine (described in Fig. 7.1) as follows. For all $\alpha \in [\ell]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c}\mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}^{(j,\beta)}_{i,b}, \mathbf{C}^{(j,\beta)}_{i,b}\right\}_{(i,j,\beta,b)\in S}, \\ \left\{\mathbf{P}_{i,v}\right\}_{(i,v)\in[\ell]\times[w]}, \\ \left\{T_i\right\}_{i\in[\ell]}, \mathsf{BP}\end{array}\right).$$

(b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows.

(a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.

(b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

$$\begin{aligned}
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
&\forall\ (j,\beta) \in [\lambda]\times\{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
&\forall\ (i,j,\beta) \in [\ell]\times[\lambda]\times\{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall\ (j,\beta) \in \mathsf{comm}\times\{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
&\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
\end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\forall\ (i,\beta,b) \in [\ell]\times\{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*,\\ \mathbf{U}_{i,b} & \text{if } \beta = 1-\beta^*. \end{cases}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\begin{aligned}
\forall\ (i,j,\beta) \in [\ell+1]\times\mathsf{comm}\times\{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} &= \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},\\
\forall\ (i,j,\beta) \in [\ell]\times\mathsf{diff}\times\{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} &= \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}.
\end{aligned}$$

For all $(j,\beta) \in \widehat{\mathsf{diff}}\times\{0,1\}$,

$$\mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell}\left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)}\prod_{\delta=\alpha}^{\ell}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)}\prod_{\delta=1}^{\ell}\mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Next, we have a sequence of ℓ hybrid experiments, $\mathsf{Game}\ 8.i^*$ for $i^* = 1$ to ℓ.

$\underline{\mathsf{Game}\ 8.i^*}$ This is identical to the previous game, except now the challenger samples the first i^* ciphertext components (both challenge and queried) as random Gaussian matrices. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

  1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

     $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j,\beta,b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

     Also, let $\widetilde{n}_i = (2\cdot|\mathsf{comm}|+w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i,j,\beta,b) \in [\ell]\times\ [\lambda] \times \{0,1\}^2\ :\ (j,\beta,b) \in S^{(i)}\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

$$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b)\in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v\in[w]} \end{bmatrix}, T_i \right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n\times m}$ for $(i,j,\beta,b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** Let

$$\mathsf{BP}^* = \left(\left\{\pi_{i,b}^* : [w] \to [w]\right\}_{i\in[\ell],b\in\{0,1\}}, \mathsf{acc}^* \in [w], \mathsf{rej}^* \in [w]\right),$$
$$S^* = \hat{S}.$$

The challenger then runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as

$$\forall\ (\alpha, b) \in [i^*] \times \{0,1\}, \quad \mathbf{U}_{\alpha,b}^* \leftarrow \chi_{\mathsf{pre}}^{m\times m}.$$

For all $\alpha \in [\ell] \setminus [i^*]$,

$$\left(\left\{\mathbf{U}_{\alpha,0}^*, \mathbf{U}_{\alpha,1}^*\right\}\right) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}^*, \alpha, S^*, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S^*}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP}^* \end{array} \right).$$

Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i\in[\ell],b\in\{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) Let $S = \hat{S}$. It runs the $\mathsf{Mixed\text{-}SubEnc}$ routine (described in Fig. 7.1) as

   $$\forall\ (\alpha, b) \in [i^*] \times \{0,1\}, \quad \mathbf{U}_{\alpha,b} \leftarrow \chi_{\mathsf{pre}}^{m \times m}.$$

   For all $\alpha \in [\ell] \setminus [i^*]$,

   $$(\{\mathbf{U}_{\alpha,0}, \mathbf{U}_{\alpha,1}\}) \leftarrow \mathsf{Mixed\text{-}SubEnc}\left(\begin{array}{c} \mathsf{tag}, \alpha, S, \\ \left\{\mathbf{B}_{i,b}^{(j,\beta)}, \mathbf{C}_{i,b}^{(j,\beta)}\right\}_{(i,j,\beta,b)\in S}, \\ \{\mathbf{P}_{i,v}\}_{(i,v)\in[\ell]\times[w]}, \\ \{T_i\}_{i\in[\ell]}, \mathsf{BP} \end{array}\right).$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.

   (b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

   $$\begin{aligned}
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m, \\
   &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m, \\
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
   &\forall\ (j,\beta) \in \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m, \\
   &\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m.
   \end{aligned}$$

(c) Let $\widetilde{x} = x^L$, and let $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

$$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^{*} & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$
$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)}.$$

For all $(j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}$,

$$\mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\big(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\big)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 9 This is identical to the previous game (i.e., Game $8.\ell$), except the last secret key components in all $\widehat{\mathsf{diff}}$ strands also include an additional noise term which is much smaller than the overall noise added in those components. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

   $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j, \beta, b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

   Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2\ :\ (j, \beta, b) \in S^{(i)}\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

   $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i, j, \beta, b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger generates ciphertext components as

  $$\forall\ (i, b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b}^* \leftarrow \chi_{\mathsf{pre}}^{m \times m}.$$

  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) It generates ciphertext components as

   $$\forall\ (i,b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b} \leftarrow \chi_{\mathsf{pre}}^{m \times m}.$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.

   (b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

   $$\begin{aligned}
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
   &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
   &\forall\ (j,\beta) \in \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
   &\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m,\\
   &\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m.
   \end{aligned}$$

   (c) Let $\widetilde{x} = x^L$, and let $\mathbf{U}_{i,b}^{(\beta)}$ denote the following:

   $$\forall\ (i,\beta,b) \in [\ell] \times \{0,1\}^2, \quad \mathbf{U}_{i,b}^{(\beta)} = \begin{cases} \mathbf{U}_{i,b}^* & \text{if } \beta = \beta^*, \\ \mathbf{U}_{i,b} & \text{if } \beta = 1 - \beta^*. \end{cases}$$

Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

$$\forall\ (i,j,\beta) \in [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$
$$\forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} = \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},$$

$$\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} \right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \cdot \prod_{\delta=2}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},$$

$$\forall\ \beta \in \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

Game 10 This is identical to the previous game, except the last secret key components in all $\widehat{\mathsf{diff}}$ strands are random vectors as well. Below we describe it in detail.

- **Setup phase.** The adversary sends the functionality index (k, w, L) and description of branching program BP^* to the challenger. Then the challenger proceeds as follows:

1. It chooses an LWE modulus q, dimensions n, m, and also distributions $\chi_{\mathsf{big}}, \chi_s, \chi_{\mathsf{appr}}, \chi_{\mathsf{pre}}, \chi_{\mathsf{last}}, \chi_{\mathsf{lwe}}$ as described in the construction. Recall that $\ell = k \cdot L$ and $\widetilde{n} = (4\lambda + w)n$. It also chooses two λ-bit strings $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$. If $\mathsf{tag}^* = \mathsf{tag}$, then it aborts and the adversary wins. Otherwise, the challenger continues as below. Let $S^{(i)}$ denote the following sets:

   $$\forall\ i \in [\ell], \quad S^{(i)} = \left\{(j, \beta, b) \in [\lambda] \times \{0,1\}^2\ :\ j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\right\}.$$

   Also, let $\widetilde{n}_i = (2 \cdot |\mathsf{comm}| + w)n$ for all $i \in [\ell]$, and set $\hat{S} = \{(i, j, \beta, b) \in [\ell] \times [\lambda] \times \{0,1\}^2\ :\ (j, \beta, b) \in S^{(i)}\}$.

2. It samples $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i,j,\beta,b}, \{\mathbf{P}_{i,v}\}_{i,v}$ matrices as

   $$\forall\ i \in [\ell], \quad \left(\begin{bmatrix} \left\{\mathbf{B}_{i,b}^{(j,\beta)}\right\}_{(j,\beta,b) \in S^{(i)}} \\ \{\mathbf{P}_{i,v}\}_{v \in [w]} \end{bmatrix}, T_i\right) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q).$$

3. It then samples matrices $\mathbf{C}_{i,b}^{(j,\beta)} \leftarrow \mathbb{Z}_q^{n \times m}$ for $(i, j, \beta, b) \in \hat{S}$.

4. Finally, it sends the public parameters $\mathsf{pp} = (\lambda, n, m, q, k, w, L, \chi_{\mathsf{pre}})$ to the adversary.

- **Challenge phase.** The challenger generates ciphertext components as

  $$\forall\ (i, b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b}^* \leftarrow \chi_{\mathsf{pre}}^{m \times m}.$$

  Finally, it sends the challenge ciphertext as $\left(\mathsf{tag}^*, \left\{\mathbf{U}_{i,b}^*\right\}_{i \in [\ell], b \in \{0,1\}}\right)$.

- **Post-challenge phase.** The adversary is allowed to make at most 1 secret key encryption query, followed by polynomially many secret key queries. The challenger responds to each query as below.

1. **Ciphertext query.** The adversary sends a branching program BP for encryption. The challenger responds as follows:

   (a) It generates ciphertext components as

   $$\forall\ (i,b) \in [\ell] \times \{0,1\}, \quad \mathbf{U}_{i,b} \leftarrow \chi_{\mathsf{pre}}^{m \times m}.$$

   (b) Finally, it sends the ciphertext as $\left(\mathsf{tag}, \{\mathbf{U}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}\right)$.

2. **Secret key queries.** The adversary queries the challenger on polynomially many messages for corresponding secret keys. For each queried string x, the challenger responds as follows:

   (a) It chooses $|\mathsf{diff}| - 1$ random vectors as $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$ for $j \in \widehat{\mathsf{diff}}$.

   (b) It then chooses vectors $\mathbf{e}_i^{(j,\beta)}, \widetilde{\mathbf{t}}_i^{(j,\beta)}, \widetilde{\mathbf{e}}_i^{(j,\beta)}$ as follows:

   $$\begin{aligned}
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \mathbf{e}_i^{(j,\beta)} \leftarrow \chi_{\mathsf{big}}^m,\\
   &\forall\ (j,\beta) \in [\lambda] \times \{0,1\}, \quad \mathbf{e}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{last}}^m,\\
   &\forall\ (i,j,\beta) \in [\ell] \times [\lambda] \times \{0,1\}, \quad \widetilde{\mathbf{t}}_i^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
   &\forall\ (j,\beta) \in \mathsf{comm} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m,\\
   &\forall\ \beta \in \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} \leftarrow \mathbb{Z}_q^m,\\
   &\forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m.
   \end{aligned}$$

   (c) Let $\widetilde{x} = x^L$. Next, it computes key vectors $\{\mathbf{t}_i^{(j,\beta)}\}_{i,j,\beta}$ as follows:

   $$\begin{aligned}
   \forall\ (i,j,\beta) \in [\ell+1] \times \mathsf{comm} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} &= \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},\\
   \forall\ (i,j,\beta) \in [\ell] \times \mathsf{diff} \times \{0,1\}, \quad \mathbf{t}_i^{(j,\beta)} &= \widetilde{\mathbf{t}}_i^{(j,\beta)} + \mathbf{e}_i^{(j,\beta)},\\
   \forall\ (j,\beta) \in \widehat{\mathsf{diff}} \times \{0,1\}, \quad \mathbf{t}_{\ell+1}^{(j,\beta)} &= \widetilde{\mathbf{t}}_{\ell+1}^{(j,\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)},
   \end{aligned}$$

$$\forall\ \beta \in \{0,1\},\quad \mathbf{t}_{\ell+1}^{(j^*,\beta)} = \widetilde{\mathbf{t}}_{\ell+1}^{(j^*,\beta)} + \mathbf{e}_{\ell+1}^{(j^*,\beta)}.$$

(d) Finally, it sends the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$.

- **Guess.** The adversary finally sends the guess γ'.

7.4.4 Indistinguishability of hybrid games in Section 7.4.3

Now we show that the hybrid experiments described in Section 7.4.3 are computationally indistinguishable. For any PPT adversary $\mathcal{A}$, let $p_{\mathcal{A},x}(\cdot)$ denote the probability that adversary $\mathcal{A}$ outputs $\gamma' = 1$ in $\mathsf{Game}\ x$.

Lemma 7.4.10. *There exists a negligible function $negl(\cdot)$ such that for any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $p_{\mathcal{A},0}(\lambda) - p_{\mathcal{A},1}(\lambda) \leq negl(\lambda)$.*

Proof. The proof of this lemma is identical to that of Lemma 7.4.1. ■

Lemma 7.4.11. *For any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, $p_{\mathcal{A},1}(\lambda) = p_{\mathcal{A},2}(\lambda)$.*

Proof. The proof of this lemma is identical to that of Lemma 7.4.2. ■

Lemma 7.4.12. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq negl(\lambda)$.*

Proof. First, let us list the differences between Game 2 and Game 3.1.1. The setup, challenge phase, and ciphertext query are handled in an identical manner. The key queries, however, are handled differently. For each key query x, the challenger outputs $\{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell]\times[\lambda]\times\{0,1\}}$ as the secret key. The components $\{\mathbf{t}_1^{(j,\beta)}\}_{j\in\mathsf{diff},\beta\in\{0,1\}}$ are computed differently in Game 2 and Game 3.1.1.

In particular, in Game 3.1.1, the challenger adds an additional error term $\widetilde{\mathbf{e}}_1^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m$ in $\mathbf{t}_1^{(j,\beta)}$.

The proof of this lemma is similar to the proof of Lemma 7.4.3, (and uses the smudging lemma, Lemma 3.1.1). Therefore, $\mathsf{Adv}_{\mathcal{A},2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.1.1}(\lambda) \leq q_{\mathsf{keys}}(\lambda) \cdot (2 \cdot |\mathsf{diff}| \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda)) \leq q_{\mathsf{keys}}(\lambda) \cdot (2\lambda \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda))$, and the lemma follows by setting $\mathrm{negl}_{3.1.1} = (2\lambda \cdot \mathrm{negl}_{\mathsf{smud}})$. ■

Lemma 7.4.13. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},3.i^*.1}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) \leq negl(\lambda)$.*

Proof. Let us first consider the differences between Game $3.i^*.1$ and Game $3.i^*.2$. The setup, challenge phase, and ciphertext query are handled in an identical manner in both games. The key generation queries are computed differently (in particular the components $\{\mathbf{t}_{i^*+1}^{(j,\beta)}\}_{j \in \mathsf{diff}, \beta \in \{0,1\}}$ in each secret key).

The proof of this lemma is similar to the proof of Lemma 7.4.4, and the main idea is to use Fact 7.4.1 to argue that $\mathbf{e}_{i^*+1}^{(j,\beta)}$ drowns $\widetilde{\mathbf{e}}_{i^*+1}^{(j,\beta)} \cdot \mathbf{U}_{i^*,\widetilde{x}_{i^*}}^{(\beta)}$ and $\widetilde{\mathbf{e}}_{i^*+1}^{(j,\beta)}$. ■

Lemma 7.4.14. *Assuming the trapdoor generation algorithms $\mathsf{LT}_{\mathsf{en}}$ satisfy the $(q, \sigma_{\mathsf{pre}})$-row removal property, for any PPT adversary $\mathcal{A}$ and $i^* \in [\ell]$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.2}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) \leq negl(\lambda)$.*

Proof. This proof is similar to the proof of Lemma 7.4.5, and we will be using the row removal property to prove it. We will first present the differences

between the two games and then discuss why the row removal property is applicable here. The exact reduction from the row removal property to indistinguishability of $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$ can be found in the proof of Lemma 7.4.5.

Differences between $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$:

1. Set $S^{(i^*)}$: In $\mathsf{Game}\ 3.i^*.2$, the challenger sets $S^{(i^*)} = [\lambda] \times \{0,1\}^2$, while in $\mathsf{Game}\ 3.i^*.3$, $S^{(i^*)} = \mathsf{comm} \times \{0,1\}^2$ ($\mathsf{tag}^*, \mathsf{tag}$ are chosen at the start of the security game, so the set diff is well defined here). Also, $\widetilde{n}_{i^*} = \widetilde{n} = (4\lambda + w)n$ in $\mathsf{Game}\ 3.i^*.2$, while $\widetilde{n}_{i^*} = \widetilde{n} - |\mathsf{diff}| \cdot 4n$ in $\mathsf{Game}\ 3.i^*.3$.

2. $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i=i^*}$ matrices: In $\mathsf{Game}\ 3.i^*.2$, the challenger chooses $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}}, 1^m, q)$, while in $\mathsf{Game}\ 3.i^*.3$, it chooses matrices $(\mathbf{M}_{i^*}, T_{i^*}) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}-|\mathsf{diff}|\cdot 4n}, 1^m, q)$. As a result, in $\mathsf{Game}\ 3.i^*.2$, it derives all matrices $\{\mathbf{B}_{i^*,b}^{(j,\beta)}\}_{(j,\beta,b)\in[\lambda]\times\{0,1\}^2}$ from $\mathbf{M}_{i^*}$. In $\mathsf{Game}\ 3.i^*.3$, the challenger chooses $\{\mathbf{B}_{i^*,b}^{(j,\beta)}\}_{j\in\mathsf{diff},b,\beta\in\{0,1\}}$ uniformly at random, while the remaining are derived from $\mathbf{M}_{i^*}$.

3. Ciphertexts: Since the set $S^{(i^*)}$ is different in both games, the challenge and query ciphertexts are constructed differently in both games. In particular, the challenge ciphertext components $(\mathbf{U}_{i^*,0}^*, \mathbf{U}_{i^*,1}^*)$ and the ciphertext query components $(\mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1})$ are computed using $\mathbf{M}_{i^*}$ and T_{i^*}, which are computed differently in $\mathsf{Game}\ 3.i^*.2$ and $\mathsf{Game}\ 3.i^*.3$.

Let us now discuss why the row removal property suffices for proving this lemma.

- Consider the four matrices $\left(\mathbf{U}^*_{i^*,0}, \mathbf{U}^*_{i^*,1}, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right)$. Fix any $j \in \mathsf{diff}$, and let $\mathsf{tag}^*_j = \beta$ and $\mathsf{tag}_j = 1 - \beta$. Then, from the definitions of $\mathbf{D}^{(j,\beta)}_b$ and $\widetilde{\mathbf{D}}^{(j,\beta)}_b$ in $\mathsf{Mixed\text{-}SubEnc}$, it follows that
  - $\mathbf{B}^{(j,\beta)}_{i^*,0} \cdot \mathbf{U}^*_{i^*,0} = \mathbf{C}^{(j,\beta)}_{i^*,0}$, and the rest are mapped to random matrices;
  - $\mathbf{B}^{(j,\beta)}_{i^*,1} \cdot \mathbf{U}^*_{i^*,1} = \mathbf{C}^{(j,\beta)}_{i^*,1}$, and the rest are mapped to random matrices;
  - $\mathbf{B}^{(j,1-\beta)}_{i^*,0} \cdot \mathbf{U}_{i^*,0} = \mathbf{C}^{(j,1-\beta)}_{i^*,0}$, and the rest are mapped to random matrices;
  - $\mathbf{B}^{(j,1-\beta)}_{i^*,1} \cdot \mathbf{U}_{i^*,1} = \mathbf{C}^{(j,1-\beta)}_{i^*,1}$, and the rest are mapped to random matrices.
- Next, note that the $\{\mathbf{C}^{(j,\beta)}_{i^*,b}\}_{j \in \mathsf{diff}, b, \beta \in \{0,1\}}$ are not used for responding to key generation queries.
- Using the above points, we can conclude that in $\mathsf{Game}\ 3.i^*.2$, for each $j \in \mathsf{diff}$, each of $\left(\mathbf{U}^*_{i^*,0}, \mathbf{U}^*_{i^*,1}, \mathbf{U}_{i^*,0}, \mathbf{U}_{i^*,1}\right)$ maps $\left[\mathbf{B}^{(j,0)}_{i^*,0} \mid \mathbf{B}^{(j,0)}_{i^*,1} \mid \mathbf{B}^{(j,1)}_{i^*,0} \mid \mathbf{B}^{(j,1)}_{i^*,1}\right]$ to a uniformly random matrix.

The proof of this lemma therefore follows using the row removal property. ■

Lemma 7.4.15. *Assuming the* $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *and* $i^* \in [\ell]$ *there exists a negligible function* $negl_{3.i^*.4}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.3}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) \leq negl_{3.i^*.4}(\lambda)$.

Proof. The proof of this lemma is similar to the proof of Lemma 7.4.6, except that it involves more hybrid experiments.

In Game $3.i^*.3$, for each key query x, for each $\beta \in \{0,1\}$ and $j \in \mathsf{diff}$, the component $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} = -\sum_{\alpha=1}^{i^*-1} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) - \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{i^*-1} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}_{i^*,\mathsf{st}_{i^*}^{(\beta)}} + \mathbf{s}_{i^*}^{(j,\beta)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\beta)} + \widetilde{\mathbf{e}}_{i^*}^{(j,\beta)}$. In Game $3.i^*.4$, $\widetilde{\mathbf{t}}_{i^*}^{(j,\beta)} \leftarrow \mathbb{Z}_q^m$. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define $q_{\mathsf{keys}} \cdot \lambda$ hybrid experiments.

Hybrid $\boldsymbol{H_{o,\hat{j},0}}$ for $\boldsymbol{o \in \{0,1,\ldots,q_{\mathsf{keys}}\}}$, $\boldsymbol{\hat{j} \in [\lambda]}$ In this hybrid, for the first o keys, for $j \in \mathsf{diff} \cap [\hat{j}]$, the $\widetilde{\mathbf{t}}_{i^*}^{(j,0)}$ components are sampled uniformly at random, while the remaining components are sampled as in Game $3.i^*.3$.

Hybrid $\boldsymbol{H_{o,\hat{j},1}}$ for $\boldsymbol{o \in \{0,1,\ldots,q_{\mathsf{keys}}\}, \hat{j} \in [\lambda]}$ In this hybrid, for all keys and $j \in \mathsf{diff}$, the $\widetilde{\mathbf{t}}_{i^*}^{(j,0)}$ components are sampled uniformly at random. For the first o queries and $j \in \mathsf{diff} \cap [\hat{j}]$, the $\widetilde{\mathbf{t}}_{i^*}^{(j,1)}$ components are sampled uniformly at random, while the remaining are sampled as in Game $3.i^*.3$.

Clearly, $H_{0,0,0}$ corresponds to Game $3.i^*.3$, $H_{q_{\mathsf{keys}},\lambda,1}$ is identical to Game $3.i^*.4$, $H_{o-1,\lambda,b} \equiv H_{o,0,b}$, and $H_{q_{\mathsf{keys}},\lambda,0} \equiv H_{0,0,1}$. Let $a_{\mathcal{A},i,\hat{j},b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{i,\hat{j},b}$.

Claim 7.4.3. *Assuming the* $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *making* $q_{\mathsf{keys}}(\cdot)$ *key queries there exists a negligible function* $\mathsf{n}_{o,\hat{j},0}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ *and all indices* $o \in [q_{\mathsf{keys}}]$ *and* $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,j-1,0} - a_{\mathcal{A},o,j,0} \leq \mathsf{n}_{o,\hat{j},0}(\lambda)$.

Claim 7.4.4. *Assuming the* $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *making* $q_{\mathsf{keys}}(\cdot)$ *key queries there exists a negligible function* $\mathsf{n}_{o,0,0}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ *and all indices* $o \in [q_{\mathsf{keys}}]$ *and* $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,\hat{j}-1,1} - a_{\mathcal{A},o,\hat{j},1} \leq \mathsf{n}_{o,\hat{j},1}(\lambda)$.

The proofs of these claims are similar to the proof of Claim 7.4.1. If $\hat{j} \notin \mathsf{diff}$, then $H_{o,\hat{j}-1,b} \equiv H_{o,\hat{j},b}$. Otherwise, we can reduce LWE to the indistinguishability of these two hybrids. ■

Lemma 7.4.16. *For any PPT adversary* $\mathcal{A}$ *and* $i^* \in [\ell-1]$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.i^*.4}(\lambda) - \mathsf{Adv}_{\mathcal{A},3.(i^*+1).1}(\lambda) \leq negl(\lambda)$.

This proof is identical to the proof of Lemma 7.4.12.

Lemma 7.4.17. *For any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},3.\ell.4}(\lambda) - \mathsf{Adv}_{\mathcal{A},4}(\lambda) \leq negl(\lambda)$.

Proof. The only difference between Game $3.\ell.4$ and Game 4 is that the $\mathbf{t}_{\ell+1}^{(j^*,\beta)}$ terms contain an additional noise term, $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta)}$. The proof of this lemma is identical to the proof of Lemma 7.4.12 and follows via the smudging lemma (Lemma 3.1.1). ■

Lemma 7.4.18. *Let* $\sigma : \mathbb{N} \to \mathbb{R}^+$ *and* $q : \mathbb{N} \to \mathbb{N}$ *be functions, and* $\chi_s(\lambda) \equiv \mathcal{D}_{\sqrt{2}\sigma(\lambda)}$ *and* $\chi_{\mathsf{lwe}}(\lambda) \equiv \mathcal{D}_{\sigma(\lambda)}$ *for each* $\lambda \in \mathbb{N}$. *Assuming the* $\mathsf{LWE\text{-}ss}_{(n,q,\sigma_{\mathsf{lwe}})}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},4}(\lambda) - \mathsf{Adv}_{\mathcal{A},5}(\lambda) \leq negl(\lambda)$.

Proof. The only difference between Game 4 and Game 5 is in the key generation phase. In Game 4, for each query, the term $\mathbf{t}_{\ell+1}^{(j^*,\beta^*)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta^*)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta^*)}\right) + \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,\beta^*)}$, and similarly the term $\mathbf{t}_{\ell+1}^{(j^*,1-\beta^*)} = \widetilde{\mathbf{s}} \cdot \mathbf{P}^{(1-\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(1-\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)} +$ other terms . In Game 5, for each key query, both $\mathbf{t}_{\ell+1}^{(j^*,0)}$ and $\mathbf{t}_{\ell+1}^{(j^*,1)}$ are set to be uniformly random. Using the short secrets version of LWE, we can switch both $\widetilde{\mathbf{s}} \cdot \mathbf{P}^{(\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)}$ and $\widetilde{\mathbf{s}} \cdot \mathbf{P}^{(1-\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(1-\beta^*)}} + \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)}$ to uniformly random vectors. This switch is possible because

- $\widetilde{\mathbf{s}}$ is chosen from χ_s^n, and $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)}, \widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)}$ are chosen from χ_{lwe}^m;
- $\widetilde{\mathbf{s}}$, $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,\beta^*)}$, and $\widetilde{\mathbf{e}}_{\ell+1}^{(j^*,1-\beta^*)}$ are not required anywhere else in Game 4 or Game 5; each of these three terms is chosen afresh for each key query;
- $\mathbf{P}^{(\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(\beta^*)}}$ and $\mathbf{P}^{(1-\beta^*)}_{\ell+1,\mathsf{st}_{\ell+1}^{(1-\beta^*)}}$ are uniformly random matrices.

Formally, we will show that Game 4 and Game 5 are computationally indistinguishable via a sequence of hybrid experiments. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define q_{keys} hybrid experiments.

Hybrid H_o for $o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}$ In this hybrid, for the first o keys, the $\mathbf{t}_{i^*+1}^{(j^*,0)}, \mathbf{t}_{lvl+1}^{(j^*,1)}$ components are sampled uniformly at random in the first o queries. For the remaining $q_{\mathsf{keys}} - o$ key queries, the keys are generated as in Game 4 in the remaining queries.

Clearly, H_0 corresponds to Game 4, while $H_{q_{\mathsf{keys}}}$ is identical to Game 5. Let $a_{\mathcal{A},i}(\lambda)$ denote the advantage of $\mathcal{A}$ in H_i.

Claim 7.4.5. *Let $\sigma : \mathbb{N} \to \mathbb{R}^+$ and $q : \mathbb{N} \to \mathbb{N}$ be functions, and $\chi_s(\lambda) \equiv \mathcal{D}_{\sqrt{2}\sigma(\lambda)}$ and $\chi_{\mathsf{lwe}}(\lambda) \equiv \mathcal{D}_{\sigma(\lambda)}$ for each $\lambda \in \mathbb{N}$. Assuming the $\mathsf{LWE\text{-}ss}_{(n,q,\sigma_{\mathsf{lwe}})}$ assumption holds, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$, $a_{\mathcal{A},o-1} - a_{\mathcal{A},o} \leq negl(\lambda)$.*

Proof. Suppose there exist an adversary making q_{keys} key queries, and a non-negligible function $\eta(\cdot)$ such that for all $\lambda \in \mathbb{N}$, there exists an index $o \in [q_{\mathsf{keys}}]$ such that $a_{\mathcal{A},o-1} - a_{\mathcal{A},o} \geq \eta(\lambda)$. We will use $\mathcal{A}$ to build a reduction algorithm $\mathcal{B}$ that breaks the $\mathsf{LWE\text{-}ss}_{(n,q,\sigma_{\mathsf{lwe}})}$ assumption.

The reduction algorithm receives (k, w, L) from the adversary and sets the parameters as in H_{o-1}/H_o. It makes $2m$ queries to the LWE-ss challenger and receives $\{(\mathbf{a}_j, u_j)\}_{j \leq 2m}$. It chooses $\mathsf{tag}^*, \mathsf{tag} \leftarrow \{0,1\}^\lambda$, and j^* is the first position where tag^* and tag differ. Next, it sets $\widetilde{n}_i$ as in H_{o-1}/H_o and chooses $(\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q)$.

Challenge phase The reduction algorithm receives challenge ciphertext BP^*, which specifies the reject state rej^*, and uses Mixed-SubEnc for computing the challenge ciphertext. Note that Mixed-SubEnc chooses $\mathbf{P}_{\ell+1,v}$ uniformly at random. The reduction algorithm sets $\mathbf{P}_{\ell+1,\mathsf{rej}^*}$ to be a matrix whose jth column is $\mathbf{a}_j^T$. All other $\mathbf{P}_{\ell+1,v}$ are chosen uniformly at random.

Ciphertext query The reduction algorithm receives ciphertext query BP and uses $\mathsf{Mixed\text{-}SubEnc}$ for computing the ciphertext query. Let rej denote the reject state of BP. It sets $\mathbf{P}_{\ell+1,\mathsf{rej}}$ to be a matrix whose jth column is $\mathbf{a}_{m+j}^T$. The remaining $\mathbf{P}_{\ell+1,v}$ matrices are chosen uniformly at random.

Key queries The reduction algorithm first sets $\mathbf{u}^{\beta^*} = [u_1 \dots u_m]$ and $\mathbf{u}^{1-\beta^*} = [u_{m+1} \dots u_{2m}]$. For the first $o-1$ key queries, the $\mathbf{t}_{\ell+1}^{(j^*,\beta)}$ components are chosen uniformly at random. For the oth key query, the reduction algorithm sets $\mathbf{t}_{\ell+1}^{(j^*,\beta^*)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,\beta^*)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta^*)}\right) + \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,\beta^*)} + \mathbf{u}^{\beta^*}$ and $\mathbf{t}_{\ell+1}^{(j^*,1-\beta^*)} = -\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j^*,1-\beta^*)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(1-\beta^*)}\right) + \widetilde{\mathbf{y}} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(1-\beta^*)} + \mathbf{e}_{\ell+1}^{(j^*,1-\beta^*)} + \mathbf{u}^{1-\beta^*}$. The remaining key queries are handled as in H_{o-1}/H_o.

Now, if all the u_j terms output by the LWE challenger are uniformly random, then $\mathbf{t}_{\ell+1}^{(j^*,\beta^*)}$ and $\mathbf{t}_{\ell+1}^{(j^*,1-\beta^*)}$ are uniformly random, and hence the reduction algorithm simulates H_o. If each $u_j = \widetilde{\mathbf{s}} \cdot \mathbf{a}_j^T + \widetilde{\mathbf{e}}_j$, then the reduction algorithm simulates H_{o-1}. ■

■

Lemma 7.4.19. *Assuming the trapdoor system* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* $(q, \sigma_{\mathsf{pre}})$*-row removal property, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},5}(\lambda) - \mathsf{Adv}_{\mathcal{A},5.1}(\lambda) \leq negl(\lambda)$.

Proof. Let us first consider the differences between Game 5 and Game 5.1.

- Set $S^{(1)}$: In Game 5, the challenger sets $S^{(1)} = \mathsf{comm} \times \{0,1\}^2$, while in Game 5.1, $S^{(1)} = \{(j,\beta,b) : j \in \mathsf{comm} \wedge \beta = \mathsf{tag}_j\}$ ($\mathsf{tag}^*, \mathsf{tag}$ are chosen

at the start of the security game, so these sets are well defined here). Also, $\widetilde{n}_1 = (4|\mathsf{comm}| + w)n$ in Game 5, while $\widetilde{n}_1 = (2|\mathsf{comm}| + w)n$ in Game 5.1.

- $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{i=1}$ matrices: In Game 5, the challenger chooses matrices $(\mathbf{M}_1, T_1) \leftarrow \mathsf{EnTrapGen}(1^{(4|\mathsf{comm}|+w)n}, 1^m, q)$, while in Game $3.i^*.3$, it chooses the matrices $(\mathbf{M}_1, T_1) \leftarrow \mathsf{EnTrapGen}(1^{(2|\mathsf{comm}|+w)n}, 1^m, q)$. As a result, in Game 5, it derives all $\{\mathbf{B}_{1,b}^{(j,\beta)}\}_{(j,\beta,b)\in\mathsf{comm}\times\{0,1\}^2}$ from $\mathbf{M}_1$. In Game 5.1, the challenger chooses $\{\mathbf{B}_{1,b}^{(j,\beta)}\}_{j\in\mathsf{comm},\beta\neq\mathsf{tag}_j,b\in\{0,1\}}$ uniformly at random, while the remaining are derived from $\mathbf{M}_1$.

- Ciphertexts: Since the set $S^{(1)}$ is different in both games, the challenge ciphertext components $(\mathbf{U}_{1,0}^*, \mathbf{U}_{1,1}^*)$ and the ciphertext query components $(\mathbf{U}_{1,0}, \mathbf{U}_{1,1})$ are computed using $\mathbf{M}_1$ and T_1, which are computed differently in Game 5 and Game 5.1.

Let us now discuss why the row removal property suffices for proving this lemma. Consider any matrix $\mathbf{U} \in \left\{\mathbf{U}_{1,0}^*, \mathbf{U}_{1,1}^*, \mathbf{U}_{1,0}, \mathbf{U}_{1,1}\right\}$, fix any $j \in \mathsf{comm}$, and let $\beta = \mathsf{tag}_j$. Then, from the definitions of $\mathbf{D}_b^{(j,\beta)}$ and $\widetilde{\mathbf{D}}_b^{(j,\beta)}$ in Mixed-SubEnc, it follows that $\mathbf{B}_{1,b}^{(j,1-\beta)} \cdot \mathbf{U}$ is a random matrix for both $b \in \{0,1\}$ (because $\mathsf{tag}_j^* = \beta$).

The proof of this lemma therefore follows using the row removal property (the reduction algorithm is similar to the one described in the proof of Lemma 7.4.5). ■

Lemma 7.4.20. *Assuming the trapdoor system* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* $(q, \sigma_{\mathsf{pre}})$*-row removal property, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $i^* \in [\ell]$ *,* $\mathsf{Adv}_{\mathcal{A},5.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},5.\ell}(\lambda) \leq negl(\lambda)$*.*

The proof of this lemma is identical to the proof of Lemma 7.4.19.

Lemma 7.4.21. *Assuming the* $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $\mathsf{Adv}_{\mathcal{A},5.\ell}(\lambda) - \mathsf{Adv}_{\mathcal{A},6.2}(\lambda) \leq negl(\lambda)$*.*

Proof. The proof of this lemma is similar to the proof of Lemma 7.4.6.

In Game $5.\ell$, for each key query x, for each $j \in \mathsf{comm}$, the components $\mathbf{t}_1^{(j,\beta)} = \mathbf{s}_1^{(j,\beta)}\mathbf{B}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\beta)}$ and $\mathbf{t}_2^{(j,\beta)} = -\mathbf{s}_1^{(j,\beta)} \cdot \mathbf{C}_{1,\widetilde{x}_1}^{(j,\beta)} + \mathbf{s}_2^{(j,\beta)} \cdot \mathbf{B}_{2,\widetilde{x}_2}^{(j,\beta)} + \mathbf{e}_2^{(j,\beta)}$. In Game 6.1, then $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}, \mathbf{t}_2^{(j,1-\mathsf{tag}_j)} \leftarrow \mathbb{Z}_q^m$. Let $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ denote the number of keys queried by $\mathcal{A}(1^\lambda)$. To prove that these two games are computationally indistinguishable, we will define $q_{\mathsf{keys}} \cdot \lambda$ hybrid experiments.

Hybrid $\boldsymbol{H_{o,\hat{j},0}}$ **for** $\boldsymbol{o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}}$**,** $\boldsymbol{\hat{j} \in [\lambda]}$ In this hybrid, for the first o keys, for $j \in \mathsf{comm} \cap [\hat{j}]$, the $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}, \mathbf{t}_2^{(j,1-\mathsf{tag}_j)}$ components are sampled uniformly at random, while the remaining components are sampled as in Game $5.\ell$.

Hybrid $\boldsymbol{H_{o,\hat{j},1}}$ **for** $\boldsymbol{o \in \{0, 1, \ldots, q_{\mathsf{keys}}\}, \hat{j} \in [\lambda]}$ This hybrid is similar to the previous one, except that for $j = \hat{j} + 1$ it adds an additional χ_{lwe} noise to

$\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}$ and $\mathbf{t}_2^{(j,1-\mathsf{tag}_j)}$.

Clearly, $H_{0,0,0}$ corresponds to $\mathsf{Game}\ 5.\ell$, $H_{q_{\mathsf{keys}},\lambda,1}$ is identical to $\mathsf{Game}\ 6.2$, and $H_{o-1,\lambda,0} \equiv H_{o,0,0}$. Let $a_{\mathcal{A},o,\hat{j},b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{o,\hat{j},b}$.

Claim 7.4.6. *For every PPT adversary $\mathcal{A}$ making q_{keys} queries, there exists a $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $o \in [q_{\mathsf{keys}}]$, and $j \in [\lambda]$, $a_{\mathcal{A},o,\hat{j},0} - a_{\mathcal{A},o,\hat{j},1} \leq negl(\lambda)$.*

The proof of this claim follows via the smudging lemma (Lemma 3.1.1).

Claim 7.4.7. *Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption holds, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$, and all indices $o \in [q_{\mathsf{keys}}]$ and $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,\hat{j}-1,1} - a_{\mathcal{A},o,\hat{j},0} \leq negl(\lambda)$.*

Proof. The proof of this claim follows from the LWE assumption. The reduction algorithm makes $4m$ queries to the LWE challenger and receives $\{\mathbf{a}_j, u_j\}_{j\in 4m}$. It sets $\mathbf{B}_{1,0}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_1^T \dots \mathbf{a}_m^T]$, $\mathbf{C}_{1,0}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_{m+1}^T \dots \mathbf{a}_{2m}^T]$, $\mathbf{B}_{1,1}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_{2m+1}^T \dots \mathbf{a}_{3m}^T]$, $\mathbf{C}_{1,1}^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = [\mathbf{a}_{3m+1}^T \dots \mathbf{a}_{4m}^T]$. It also sets $\mathbf{u}_{1,0} = [u_1 \dots u_m]$, $\mathbf{u}_{2,0} = [u_{m+1} \dots u_{2m}]$, $\mathbf{u}_{1,1} = [u_{2m+1} \dots u_{3m}]$, $\mathbf{u}_{2,1} = [u_{3m+1} \dots u_{4m}]$.

For the oth key query x, the reduction algorithm sets $\mathbf{t}_1^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = \mathbf{u}_{1,\widetilde{x}_1} + \mathbf{y}^{(\hat{j})} + \mathbf{e}_1^{(\hat{j},1-\mathsf{tag}_{\hat{j}})}$ and $\mathbf{t}_2^{(\hat{j},1-\mathsf{tag}_{\hat{j}})} = \mathbf{u}_{2,\widetilde{x}_1} + \mathbf{e}_2^{(\hat{j},1-\mathsf{tag}_{\hat{j}})}$. The rest of the key components can be handled without the LWE challenge terms. ■

■

Lemma 7.4.22. *Assuming the* $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $i^* \in \{3, \ldots, \ell\}$, $\mathsf{Adv}_{\mathcal{A},6.i^*-1}(\lambda) - \mathsf{Adv}_{\mathcal{A},6.i^*}(\lambda) \leq negl(\lambda)$.

Proof. The only difference between Game $6.(i^* - 1)$ and Game $6.i^*$ is with respect to the key queries. In Game $6.(i^*-1)$, for each $j \in \mathsf{comm}$, the challenger sets $\mathbf{t}_{i^*}^{(j,1-\mathsf{tag}_j)} = -\mathbf{s}_{i^*-1}^{(j,1-\mathsf{tag}_j)} \cdot \mathbf{C}_{i^*-1,\widetilde{x}_{i-1}}^{(j,1-\mathsf{tag}_j)} + \mathbf{s}_{i^*}^{(j,1-\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_i}^{(j,1-\mathsf{tag}_j)} + \mathbf{e}_{i^*}^{(j,1-\mathsf{tag}_j)}$, while it switches these terms to random in Game $6.i^*$.

The proof of this lemma uses (standard) LWE, similar to the proof of Lemma 7.4.21. One minor difference between this lemma and Lemma 7.4.21 is that in the previous lemma, the challenger switches both $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}$ and $\mathbf{t}_2^{(j,1-\mathsf{tag}_j)}$ (this is because the vector $\mathbf{s}_1^{(j,1-\mathsf{tag}_j)}$ is used for computing both of these components). However, in this lemma, the reduction algorithm sets the LWE challenge's public vectors as the rows of $\mathbf{C}_{i^*,0}^{(j,1-\mathsf{tag}_j)}$ and $\mathbf{C}_{i^*,1}^{(j,1-\mathsf{tag}_j)}$, and the LWE challenge is used for setting $\mathbf{t}_{i^*}^{(j,1-\mathsf{tag}_j)}$. ■

Lemma 7.4.23. *For any adversary* $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},6.(\ell+1)}(\lambda) = \mathsf{Adv}_{\mathcal{A},7.1}(\lambda)$.

Proof. The only difference between Game $6.(\ell+1)$ and Game 7.1 is with respect to the $\{\mathbf{t}_1^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ components in key queries. In Game 6, for each key query x, the challenger chooses $\{\mathbf{y}^{(j)}\}_{j\neq j^*}$ and sets $\mathbf{t}_1^{(j,\mathsf{tag}_j)} = \mathbf{s}_1^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{1,\widetilde{x}_1}^{(j,\mathsf{tag}_j)} + \mathbf{y}^{(j)} + \mathbf{e}_1^{(j,\mathsf{tag}_j)}$ for each $j \in \mathsf{comm}$. In Game 7.1, the $\{\mathbf{t}_1^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ vectors are set to be uniformly random vectors.

Note that in Game $6.(\ell + 1)$, the $\mathbf{y}^{(j)}$ terms are chosen afresh for each key, and $\mathbf{y}^{(j)}$ is only used in constructing $\mathbf{t}_1^{(j,\mathsf{tag}_j)}$ (recall $\mathbf{t}_1^{(j,1-\mathsf{tag}_j)}$ is uniformly

random). As a result, the components $\{\mathbf{t}_1^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ are uniformly random vectors, and therefore the secret keys in the two games are identically distributed. ■

Lemma 7.4.24. *Assuming the* $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ *assumption holds, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $i^* \in \{2,\ldots,\ell\}$, $\mathsf{Adv}_{\mathcal{A},7.i^*-1}(\lambda) - \mathsf{Adv}_{\mathcal{A},7.i^*}(\lambda) \leq negl(\lambda)$.

Proof. The only difference between Game $7.(i^*-1)$ and Game $7.i^*$ is with respect to the $\{\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ components in key queries. In Game $7.(i^*-1)$, for each key query x, the challenger sets $\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)} = -\mathbf{s}_{i^*-1}^{(j,\mathsf{tag}_j)} \cdot \mathbf{C}_{i^*,\widetilde{x}_{i^*-1}}^{(j,\mathsf{tag}_j)} + \mathbf{s}_{i^*}^{(j,\mathsf{tag}_j)} \cdot \mathbf{B}_{i^*,\widetilde{x}_{i^*}}^{(j,\mathsf{tag}_j)} + \mathbf{e}_{i^*}^{(j,\mathsf{tag}_j)}$ for each $j \in \mathsf{comm}$. In Game $7.i^*$, the $\{\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}\}_{j\in\mathsf{comm}}$ vectors are set to be uniformly random vectors. We will show that these two games are computationally indistinguishable via a hybrid argument. First, we will define $q_{\mathsf{keys}} \cdot \lambda$ hybrid experiments.

Hybrid $\boldsymbol{H_{o,\hat{j},0}}$ for $\boldsymbol{o \in \{0,1,\ldots,q_{\mathsf{keys}}\}}$, $\boldsymbol{\hat{j} \in [\lambda]}$ In this hybrid, for the first o keys, for $j \in \mathsf{comm} \cap [\hat{j}]$, the $\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}$ components are sampled uniformly at random, while the remaining components are sampled as in Game $7.i^*$.

Hybrid $\boldsymbol{H_{o,\hat{j},1}}$ for $\boldsymbol{o \in \{0,1,\ldots,q_{\mathsf{keys}}\}, \hat{j} \in [\lambda]}$ This hybrid is similar to the previous one, except that for $j = \hat{j}+1$ it adds an additional χ_{lwe} noise to $\mathbf{t}_{i^*}^{(j,\mathsf{tag}_j)}$.

Clearly, $H_{0,0,0}$ corresponds to Game 7.$(i^* - 1)$, $H_{q_{\mathsf{keys}},\lambda,1}$ is identical to Game 7.i^*, and $H_{o-1,\lambda,0} \equiv H_{o,0,0}$. Let $a_{\mathcal{A},o,\hat{j},b}(\lambda)$ denote the advantage of $\mathcal{A}$ in $H_{o,\hat{j},b}$.

Claim 7.4.8. *For every PPT adversary $\mathcal{A}$ making q_{keys} queries, there exists a $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $o \in [q_{\mathsf{keys}}]$ and $j \in [\lambda]$, $a_{\mathcal{A},o,\hat{j},0} - a_{\mathcal{A},o,\hat{j},1} \leq negl(\lambda)$.*

The proof of this claim follows via the smudging lemma (Lemma 3.1.1).

Claim 7.4.9. *Assuming the $\mathsf{LWE}_{n,q,\sigma_{\mathsf{lwe}}}$ assumption holds, for any PPT adversary $\mathcal{A}$ making $q_{\mathsf{keys}}(\cdot)$ key queries there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $q_{\mathsf{keys}} = q_{\mathsf{keys}}(\lambda)$ and all indices $o \in [q_{\mathsf{keys}}]$ and $\hat{j} \in [\lambda]$, $a_{\mathcal{A},o,\hat{j}-1,1} - a_{\mathcal{A},o,\hat{j},0} \leq negl(\lambda)$.*

Proof. The proof of this claim follows from the LWE assumption, where the reduction algorithm sets the LWE public vectors to be columns of $\mathbf{C}_{i^*,0}^{(\hat{j},\mathsf{tag}_{\hat{j}})}, \mathbf{C}_{i^*,1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$, and the LWE challenge is used to set $\mathbf{t}_{i^*}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$. Note that the vector $\mathbf{s}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ is used only for defining $\mathbf{t}_{i^*}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$. This is because this vector is chosen afresh for each key query, and in hybrids $H_{o,\hat{j}-1,1}/H_{o,\hat{j},0}$, the key component $\mathbf{t}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ is already random.[4] ∎

∎

Recall that Game 8 is identical to Game 7.$(\ell + 1)$.

[4] If $\mathbf{t}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ was not already switched to random, then $\mathbf{s}_{i^*-1}^{(\hat{j},\mathsf{tag}_{\hat{j}})}$ would have been used to define it.

Lemma 7.4.25. *Assuming* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* $(q, \chi_{\mathsf{appr}}, \sigma_{\mathsf{pre}})$*-target switching property and* $(q, \sigma_{\mathsf{pre}})$*-well-sampledness of preimage, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda) \leq negl(\lambda)$.

Proof. First, let us discuss the differences between Game $8.(i^*-1)$ and Game $8.i^*$. In Game $8.(i^*-1)$, the challenge ciphertext components $\{\mathbf{U}^*_{i,b}\}_{i \geq i^*, b \in \{0,1\}}$ and query ciphertext components $\{\mathbf{U}_{i,b}\}_{i \geq i^*, b \in \{0,1\}}$ are computed using Mixed-SubEnc, while the remaining are chosen from Gaussian with parameter χ_{pre}. Game $8.i^*$ is similar to Game $8.(i^*-1)$, except for the challenge ciphertext components $\mathbf{U}^*_{i^*,0}, \mathbf{U}^*_{1,1}$ and the query ciphertext components $\mathbf{U}_{1,0}, \mathbf{U}_{1,1}$ are chosen from the Gaussian distribution with parameter σ_{pre}. To show that these games are indistinguishable, we will define a hybrid experiment H.

Hybrid H This hybrid is similar to Game $8.(i^*-1)$, except that the challenger computes $\{\mathbf{U}^*_{i^*,b}, \mathbf{U}_{i^*,b}\}_{b \in \{0,1\}}$ such that they map $\mathbf{M}_{i^*}$ to uniformly random matrices.

Let $\mathsf{Adv}_{\mathcal{A},H}$ denote the advantage of adversary $\mathcal{A}$ in hybrid H.

Claim 7.4.10. *Assuming* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* $(q, \chi_{\mathsf{appr}}, \sigma_{\mathsf{pre}})$*-target switching property, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $i^* \in [\ell]$, $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},H}(\lambda) \leq negl(\lambda)$.

Proof. The proof of this claim is similar to the proof of Lemma 7.4.8. First, let us discuss the reasons why the target switching property is applicable here. Let

$S^{(i^*)}$ be defined as in $\mathsf{Game}\ 8.(i^*-1)$/$\mathsf{Game}\ 8.i^*$, with $\mathsf{BP}^* = \{\pi^*_{i,b}\}_{(i,b)\in[\ell]\times\{0,1\}}$ and $\mathsf{BP} = \{\pi_{i,b}\}_{(i,b)\in[\ell]\times\{0,1\}}$ the challenge/query programs.

1. In both $\mathsf{Game}\ 8.(i^*-1)$ and hybrid H, the components $\left\{\mathbf{U}^*_{i,b}, \mathbf{U}_{i,b}\right\}$ are chosen from a Gaussian distribution, and therefore these terms do not contain any information about the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ or $\{\mathbf{B}^{(j,\beta)}_{i^*,b}\}_{(j,\beta,b)\in S^{(i^*)}}$ matrices.

2. The components $\left\{\mathbf{U}^*_{i,b}, \mathbf{U}_{i,b}\right\}_{i>i^*,b\in\{0,1\}}$ do not contain any information about the $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ or $\{\mathbf{B}^{(j,\beta)}_{i^*,b}\}_{(j,\beta,b)\in S^{(i^*)}}$ matrices (this follows from the construction).

3. The keys are all either random vectors or computed in terms of the challenge/query ciphertext components, and therefore do not explicitly require $\{\mathbf{P}_{i^*,v}\}_{v\in[w]}$ or $\{\mathbf{B}^{(j,\beta)}_{i^*,b}\}_{(j,\beta,b)\in S^{(i^*)}}$ matrices.

Consider matrices $\{\mathbf{Z}^*_{0,b}, \mathbf{Z}^*_{1,b}, \mathbf{Z}_{0,b}, \mathbf{Z}_{1,b}\}_{b\in\{0,1\}}$ defined as follows:

$$\mathbf{Z}^*_{0,b} = \begin{bmatrix} \left\{\mathbf{C}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b)\in S^{(i^*)}} \\ \left\{\mathbf{P}_{i^*,\pi^*_{i^*,b}(v)}\right\}_{v\in[w]} \end{bmatrix} \mathbf{Z}^*_{1,b} = \left[\leftarrow \mathbb{Z}_q^{\widetilde{n}_i\times m}\right],$$

$$\mathbf{Z}_{0,b} = \begin{bmatrix} \left\{\mathbf{C}^{(j,\beta)}_{i^*,b}\right\}_{(j,\beta,b)\in S^{(i^*)}} \\ \left\{\mathbf{P}_{i^*,\pi_{i^*,b}(v)}\right\}_{v\in[w]} \end{bmatrix} \mathbf{Z}_{1,b} = \left[\leftarrow \mathbb{Z}_q^{\widetilde{n}_i\times m}\right].$$

The reduction algorithm sends $\left(1^{\widetilde{n}_i}, 1^m, \emptyset\right)$ to the target switching property challenger.[5] It does not receive any matrix from the challenger (since the challenge set is empty). Next, it chooses $(M_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q)$ for all

[5]Note that the set specified by the adversary in the target switching property game can be empty.

$i > i^*$, and parses $\mathbf{M}_i$ as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ to obtain $\{\mathbf{B}_{i,b}^{(j,\beta)}\}_{(j,\beta,b)\in S^{(i)}}$ and $\{\mathbf{P}_{i,v}\}_{v\in[w]}$ for all $i > i^*$. It receives BP^* as the challenge query from the adversary. The reduction algorithm sends $\mathbf{Z}_{0,b}^*, \mathbf{Z}_{1,b}^*$ to the target switching property challenger and receives $\mathbf{U}_{i^*,b}$ in response. It chooses the remaining components as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ and sends the challenge ciphertext to the adversary.

Next, it receives the ciphertext query BP. It sends $\mathbf{Z}_{0,b}, \mathbf{Z}_{1,b}$ to the target switching property challenger and receives $\mathbf{U}_{i^*,b}$. The remaining ciphertext components are chosen as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$, and the reduction algorithm sends the challenge ciphertext to the adversary. Finally, the adversary makes key queries. For each key query, the reduction algorithm sets $\{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}$ as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ and sends them to the adversary. The adversary sends its guess, which the reduction algorithm forwards to the challenger.

Therefore, if there exist a PPT adversary $\mathcal{A}$ and a nonnegligible function η such that $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},H}(\lambda) \geq \eta(\lambda)$ for all λ, then there exists a PPT algorithm $\mathcal{B}$ that breaks the target switching property. ■

Claim 7.4.11. *Assuming* $\mathsf{LT}_{\mathsf{en}}$ *satisfies the* $(q, \sigma_{\mathsf{pre}})$*-well-sampledness of the preimage, for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},H}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda) \leq negl(\lambda)$.

Proof. This proof follows directly from the (q, σ)-well-sampledness of the preimage property. Suppose there exist a PPT adversary $\mathcal{A}$ and a reduction

algorithm $\eta(\cdot)$ such that $\mathsf{Adv}_{\mathcal{A},H}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda) \geq \eta(\lambda)$ for all $\lambda \in \mathbb{N}$. Then there exists a reduction algorithm that breaks the (q,σ)-well-sampledness of the preimage property.

The reduction algorithm sends $1^{\widetilde{n}_i}, 1^m, 1^{4m}$ to the challenger. Note that $m > \widetilde{n}_i \log q + \lambda$ and $\sigma > \sqrt{n \cdot \log q \cdot \log m} + \lambda$, as required. It receives a matrix $\mathbf{U} \in Z_q^{m \times 4m}$, which it parses as $\mathbf{U} = \left[\mathbf{U}^*_{i^*,0} \mid \mathbf{U}^*_{i^*,1} \mid \mathbf{U}_{i^*,0} \mid \mathbf{U}_{i^*,1}\right]$. The reduction algorithm also chooses $(\mathbf{M}_i, T_i) \leftarrow \mathsf{EnTrapGen}(1^{\widetilde{n}_i}, 1^m, q)$ for all $i \neq i^*$.

On receiving the challenge ciphertext, it chooses the remaining ciphertext components as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$ and sends them to the adversary. Similarly, it handles the ciphertext query. Finally, for the key queries, the reduction algorithm handles them as in $\mathsf{Game}\ 8.(i^*-1)/\mathsf{Game}\ 8.i^*$. ■

Using these two claims, it follows that $\mathsf{Adv}_{\mathcal{A},8.(i^*-1)}(\lambda) - \mathsf{Adv}_{\mathcal{A},8.i^*}(\lambda)$ is bounded by a negligible function. ■

Lemma 7.4.26. *For any PPT adversary $\mathcal{A}$ there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},8.\ell}(\lambda) - \mathsf{Adv}_{\mathcal{A},9}(\lambda) \leq negl(\lambda)$.*

Proof. The only difference between these two hybrids is with respect to the key components $\{\mathbf{t}_\ell^{(j,\beta)}\}_{j \in \widehat{\mathsf{diff}}, \beta \in \{0,1\}}$. In $\mathsf{Game}\ 8.\ell$, these vectors are computed as $\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}$, while in $\mathsf{Game}\ 9$, this vector is set to be $\sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_\alpha^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \mathbf{y}^{(j)} \cdot \prod_{\delta=1}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \cdot \prod_{\delta=2}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}$, where $\widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m$. Note that the $\mathbf{U}_{i,b}$ matrices are all drawn from the

Gaussian distribution with parameter σ_{pre}, and therefore, with all but negligible probability, $\left\|\widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \cdot \prod_{\delta=2}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right\| \leq m\sigma_{\mathsf{lwe}} \cdot (m\sigma_{\mathsf{pre}})^{\ell-1}$. Since $\sigma_{\mathsf{last}}/m\sigma_{\mathsf{lwe}} \cdot (m\sigma_{\mathsf{pre}})^{\ell-1} \geq 2^{\lambda}$, we can use the smudging lemma to argue that the statistical distance between these two games is at most $m \cdot q_{\mathsf{keys}} \cdot \mathrm{negl}_{\mathsf{smud}}(\lambda)$. ■

Lemma 7.4.27. *Assuming the* $\mathsf{LWE\text{-}sp}_{d,q,\sigma_{\mathsf{pre}},\chi_{\mathsf{lwe}}}$ *assumption* (Assumption 3) *holds (where* $d(\lambda) = 6n \cdot \log q$*), for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathit{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\mathsf{Adv}_{\mathcal{A},9}(\lambda) - \mathsf{Adv}_{\mathcal{A},10}(\lambda) \leq \mathit{negl}(\lambda)$.

Proof. The only difference between Game 9 and Game 10 is with respect to the key components $\{\mathbf{t}_{\ell+1}^{(j,\beta)}\}_{j\in\widehat{\mathsf{diff}},\beta\in\{0,1\}}$. In Game 9, for each key, these are computed as $\mathbf{t}_{\ell+1}^{(j,\beta)} = \sum_{\alpha=1}^{\ell} \left(\widetilde{\mathbf{t}}_{\alpha}^{(j,\beta)} \cdot \prod_{\delta=\alpha}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)}\right) + \left(\mathbf{y}^{(j)} \cdot \mathbf{U}_{1,\widetilde{x}_1}^{(\beta)} + \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)}\right) \prod_{\delta=2}^{\ell} \mathbf{U}_{\delta,\widetilde{x}_\delta}^{(\beta)} + \mathbf{e}_{\ell+1}^{(j,\beta)}$, while in Game 10, these vectors are uniformly random. To prove this claim, it suffices to switch $\left(\mathbf{y}^{(j)} \cdot \mathbf{U}_{1,\widetilde{x}_1}^{(\beta)} + \widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)}\right)$ to a uniformly random vector. Since $\mathbf{y}^{(j)} \leftarrow \mathbb{Z}_q^m$, $\mathbf{U}_{1,\widetilde{x}_1}^{(\beta)} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{pre}}}^{m\times m}$, and $\widetilde{\mathbf{e}}_{\ell+1}^{(j,\beta)} \leftarrow \chi_{\mathsf{lwe}}^m$, we can use the LWE-sp assumption as in the proof of Theorem 6.3.4. ■

7.4.5 Proving 1-query restricted accept indistinguishability

First, note that by using the lemmas provided in Section 7.4.4, we can conclude that our construction satisfies 1-query complete accept indistinguishability security. Concretely, algorithms $\mathsf{SK\text{-}Enc}^*$ and KeyGen^*, which take as input the public parameters, are defined as follows: $\mathsf{SK\text{-}Enc}^*$ is the same as the standard encryption algorithm Enc (that is, it outputs random Gaussian

matrices), and KeyGen^* on input x outputs the secret key as $\left(x, \{\mathbf{t}_i^{(j,\beta)}\}_{(i,j,\beta)\in[\ell+1]\times[\lambda]\times\{0,1\}}\right)$, where all $\mathbf{t}_i^{(j,\beta)}$ are sampled uniformly at random from $\mathbb{Z}_q^m$. Since in Game 10 the challenger is already using $\mathsf{SK\text{-}Enc}^*$ and KeyGen^* to answer corresponding queries, therefore, using lemmas in Section 7.4.4, we can argue 1-query complete accept indistinguishability security.

Lastly, to finish the proof we only need to argue that the probability adversary outputs 1 in the 1-query *restricted* accept indistinguishability security game—when the challenger computes challenge ciphertext as a normal functional encryption ciphertext instead of encrypting BP^*—is negligibly close to the probability adversary outputs 1 in Game 10. Note that this again follows from the lemmas in Section 7.4.4, or, in other words, it follows from the fact that our construction satisfies 1-query complete accept indistinguishability security. This is because if an adversary can distinguish Game 10 from the scenario described above, then we could come up with a reduction algorithm that breaks the 1-query complete accept indistinguishability security of our construction.

The idea is straightforward. The reduction algorithm will simply forward messages (back and forth) between the attacker and the complete accept indistinguishability challenger, except with the following changes:

- The reduction algorithm does not forward the adversary's challenge program BP^* as its challenge query to the challenger of the complete accept indistinguishability game. Instead it runs the normal encryption algo-

rithm Enc and sends the output back to the adversary as its challenge ciphertext.

- In addition, the reduction algorithm sends the adversary's post-challenge encryption query (if any) to the challenger as its challenge query and forwards the challenger's response to the adversary.
- Also, the reduction algorithm does not make any post-challenge encryption query. (Note that key queries are answered as before.)
- Finally, it outputs whatever the adversary outputs.

Clearly the reduction algorithm perfectly simulates the indistinguishability experiment (between Game 10 and the scenario described above); thus if the adversary's advantage is nonnegligible, then the reduction algorithm also breaks 1-query complete accept indistinguishability security with nonnegligible probability. This completes the proof.

Chapter 8

Embedding Identities in Traitor Tracing

In this chapter, we describe how to embed identities in a traitor tracing system. We give new constructions for traitor tracing systems with embedded identity tracing under the learning with errors assumption.

We start by formally defining the concept of traitor tracing systems with embedded identity tracing. We refer to such systems as embedded identity traitor tracing (EITT) systems. In order to capture a broader class of traitor tracing systems, we consider three different variants for embedded identity tracing — (1) indexed EITT, (2) bounded EITT, and (3) full (unbounded) EITT. Here the notion of full/unbounded EITT is most general, and we show that a bounded EITT scheme (with suitable parameters) can be generically transformed into an unbounded EITT scheme, and we also show that an indexed EITT scheme directly gives a bounded EITT scheme.

Next, we move on to realizing these EITT systems under LWE. To that end, we first introduce a new intermediate primitive which we call *embedded-identity private linear broadcast encryption* (EIPLBE) that we eventually use to build EITT schemes. As the name suggests, the notion of EIPLBE is inspired by and is an extension of *private linear broadcast encryption* (PLBE)

schemes Section 5.1. Finally, we describe how to adapt the mixed FE framework developed in Section 4.2 to instantiate ans EIPLBE scheme.

8.1 Defining Embedded Identity Traitor Tracing

8.1.1 Indexed Embedded-Identity Traitor Tracing

In this section, we present the syntax and definitions for traitor tracing with embedded identities where the number of users is bounded, and the key generation is 'indexed'.

Let $\mathcal{T}$ be a (indexed keygen, public/private)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$. It consists of five algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\text{indx}}) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm takes as input the security parameter λ, the 'identity space' parameter κ, index space $[n_{\text{indx}}]$, and outputs a master secret key msk, a public key pk, and a tracing key key.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa, i \in [n_{\text{indx}}]) \to \mathsf{sk}_{i,\mathsf{id}}$. The key generation algorithm takes as input the master secret key, identity $\mathsf{id} \in \{0,1\}^\kappa$ and index $i \in [n_{\text{indx}}]$. It outputs a secret key $\mathsf{sk}_{i,\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pk}, m \in \mathcal{M}_\lambda) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1) \to T \subseteq \{0,1\}^\kappa$. The trace algorithm has oracle access to a program D, it takes as input key (which is the master secret key msk in a private-key tracing scheme, and the public key pk in a public tracing algorithm), parameter y and two messages m_0, m_1. It outputs a set T of index-identity pairs, where $T \subseteq \{0,1\}^\kappa$.

Correctness A traitor tracing scheme is said to be correct if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda, \kappa, n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, identity $\mathsf{id} \in \{0,1\}^\kappa$ and $i \in [n]$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n); \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, i); \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Efficiency Let $\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$ be functions. A (indexed keygen, public/private)-embedded identity tracing scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)- efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{indx}})$ is at most $\mathsf{T\text{-}s}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa, n_{\mathrm{indx}})$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa, n_{\mathrm{indx}})$.

- The number of oracle calls made by $\mathsf{Trace}^{D}(\mathsf{key}, 1^{y}, m_0, m_1)$ to decoding box D is at most $\mathsf{T\text{-}t}(\lambda, \kappa, n_{\text{indx}}, y)$.
- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa, n_{\text{indx}})$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa, n_{\text{indx}})$.

Security As in the traditional traitor tracing definitions, we have two security definitions. The first security definition (IND-CPA security) states that any PPT adversary should not distinguish between encryptions of different messages. This definition is identical to the IND-CPA definition in traditional traitor tracing. The second definition states that if there exists a pirate decoder box, then the tracing algorithm can trace the identity of at least one of the secret keys used to build the decoding box, and there are no 'false-positives'.

Definition 8.1.1 (IND-CPA security)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (indexed keygen, public/private)-embedded identity tracing scheme. This scheme is IND-CPA secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\mathsf{ct}) = b \;:\; \begin{array}{c} (1^{\kappa}, 1^{n_{\text{indx}}}) \leftarrow \mathcal{A}(1^{\lambda}); \\ (\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\kappa}, n_{\text{indx}}); \\ b \leftarrow \{0,1\}; (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}); \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array}\right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

Definition 8.1.2 (Secure tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (indexed keygen, public/private)-embedded identity tracing scheme. For any non-negligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Fig. 8.1.

Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}index}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^\kappa, 1^{n_{\mathrm{indx}}} \leftarrow \mathcal{A}(1^\lambda)$
- $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{indx}})$
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{key}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$

Each oracle query made by the adversary $\mathcal{A}$ consists of an index-identity pair $(i, \mathsf{id}) \in [n_{\mathrm{indx}}] \times \{0,1\}^\kappa$. Let $S_{\mathcal{TD}}$ the set of identities queried by $\mathcal{A}$. Here, oracle $O(\cdot)$ has msk hardwired and on query (i, id) it outputs $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, i)$ if index i is distinct from all previous queries made by the adversary, otherwise it outputs $\perp$. *In other words, for each index* $i \in [n_{\mathrm{indx}}]$*, the adversary is allowed to make at most one key query. However, for different indices* $i, i' \in [n_{\mathrm{indx}}]$*, the identity can be same (that is,* (i, id) *and* (i', id) *are valid queries if* $i \neq i'$*).*

Figure 8.1: Experiment Expt-TT-emb-index

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$

  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder}]$.

- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathcal{TD}}$

  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S_{\mathcal{TD}}$

  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, poly-

nomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathrm{negl}_1(\cdot)$, $\mathrm{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathrm{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}_2(\lambda).$$

Remark 8.1.1. We want to point out that in both IND-CPA and secure tracing games we require the adversary to output the index bound n_{indx} *in unary instead of binary* (i.e., $\mathcal{A}$ outputs $(1^\kappa, 1^{n_{\mathrm{indx}}})$ instead of $(1^\kappa, n_{\mathrm{indx}})$). Now since the running time of the adversary $\mathcal{A}$ is bounded by a polynomial, thus it can only select a polynomially-bounded value for index bound n_{indx}. However, the setup algorithm is given the input n_{indx} *in binary*. This distinction will later be useful in our constructions and security proofs.

8.1.2 Bounded Embedded-Identity Traitor Tracing

We now present the syntax and definitions for traitor tracing with embedded identities where the key generation is *not* indexed, but the number of key queries is bounded.

Let $\mathcal{T}$ be a (bounded keys, public/private tracing)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$. It consists of five algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}}) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm takes as input the security parameter λ, identity space index κ, bound on number of key

queries n_{bd}, and outputs a master secret key msk and a public key pk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa) \to \mathsf{sk}_{\mathsf{id}}$. The key generation algorithm takes as input the master secret key and identity $\mathsf{id} \in \{0,1\}^\kappa$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

The syntax of Enc, Dec, and Trace is identical to that for indexed EITT systems.

Correctness A traitor tracing scheme is said to be correct if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda, \kappa, n_{\mathrm{bd}} \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$ and identity $\mathsf{id} \in \{0,1\}^\kappa$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},\mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk},\mathsf{pk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}}); \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk},\mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Efficiency Let $\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$ be functions. A (bounded keys, public/private tracing)-embedded identity tracing scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)- efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}})$ is at most $\mathsf{T\text{-}s}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The number of oracle calls made by $\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1)$ to decoding box D is at most $\mathsf{T\text{-}t}(\lambda, \kappa, n_{\mathrm{bd}}, y)$.

- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa, n_{\mathrm{bd}})$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa, n_{\mathrm{bd}})$.

Security The IND-CPA security definition is identical to the one in Definition 8.1.1; the tracing-based security definition is very similar, but instead of requiring that the index queries are distinct, we require that the number of queries in the 'correct-trace experiment' is at most n_{bd}.

Definition 8.1.3 (Secure tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (bounded keys, public/private tracing)-embedded identity tracing scheme. For any non-negligible function $\epsilon(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}bd}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Fig. 8.2.

Experiment $\mathsf{Expt\text{-}TT\text{-}emb\text{-}bd}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$

- $1^{\kappa}, 1^{n_{\mathrm{bd}}} \leftarrow \mathcal{A}(1^{\lambda})$.
- $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\kappa}, n_{\mathrm{bd}})$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$.
- $T \leftarrow \mathsf{Trace}^{D}(\mathsf{key}, 1^{1/\epsilon(\lambda)}, m_0, m_1)$.

Let $S_{\mathcal{TD}}$ be the set of identities queried by $\mathcal{A}$. Here, $O(\cdot)$ is an oracle that has msk hardwired, takes as input an identity $\mathsf{id} \in \{0,1\}^{\kappa}$ and outputs $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$.

Figure 8.2: Experiment Expt-TT-emb-bd

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$

  $\mathsf{Admissible\text{-}Adv}$: $\mathcal{A}$ makes at most n_{bd} key queries

  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}]$.

- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathcal{ID}}$

  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S_{\mathcal{ID}}$

  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A traitor tracing scheme $\mathcal{T}$ is said to be ind-secure if for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathrm{negl}_1(\cdot)$, $\mathrm{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathrm{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}_2(\lambda).$$

8.1.3 Unbounded (Full) Embedded-Identity Traitor Tracing

We now present the syntax and definitions for general unbounded traitor tracing with embedded identities.

Let $\mathcal{T}$ be a (unbounded keys, public/private tracing)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$. It consists of five algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ and Trace with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm takes as input the security parameter λ, identity space index κ and outputs a master secret key msk and a public key pk.

The syntax of KeyGen, Enc, and Dec is identical to that for bounded EITT systems.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, Q_{\mathrm{bd}}, m_0, m_1) \to T \subseteq \{0,1\}^\kappa$. The trace algorithm has oracle access to a program D, it takes as input a master secret key key, parameters y and Q_{bd}, and two messages m_0, m_1. It outputs a set T of identities, where $T \subseteq \{0,1\}^\kappa$.

Correctness A traitor tracing scheme is said to be correct if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda, \kappa \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$ and identity $\mathsf{id} \in \{0,1\}^\kappa$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa); \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Efficiency Let $\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$ be functions. A (unbounded keys, public/private tracing)-embedded identity tracing scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)- efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa)$ is at most $\mathsf{T\text{-}s}(\lambda, \kappa)$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda, \kappa)$.

- The running time of $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{T\text{-}k}(\lambda, \kappa)$.
- The running time of $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda, \kappa)$.
- The number of oracle calls made by $\mathsf{Trace}^D(\mathsf{key}, 1^y, Q_{\mathrm{bd}}, m_0, m_1)$ to decoding box D is at most $\mathsf{T\text{-}t}(\lambda, \kappa, y, Q_{\mathrm{bd}})$.
- The size of the ciphertext output by $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{S\text{-}c}(\lambda, \kappa)$.
- The size of the key output by $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is at most $\mathsf{S\text{-}k}(\lambda, \kappa)$.

Security The IND-CPA security definition is identical to the one in previous sections; the tracing-based security definition is very similar, but there is no bound on the number of secret key queries.

Definition 8.1.4 (Secure tracing)**.** Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a (unbounded keys, public/private tracing)-embedded identity tracing scheme. For any non-negligible function $\epsilon(\cdot)$, polynomial $p(\cdot)$ and PPT adversary $\mathcal{A}$, consider the experiment $\mathsf{Expt\text{-}TT\text{-}emb}^{\mathcal{T}}_{\mathcal{A},\epsilon}(\lambda)$ defined in Fig. 8.3.

Experiment $\mathsf{Expt\text{-}TT\text{-}emb}^{\mathcal{T}}_{\mathcal{A},\epsilon,p}(\lambda)$

- $1^\kappa \leftarrow \mathcal{A}(1^\lambda)$.
- $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{pk})$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{key}, 1^{1/\epsilon(\lambda)}, p(\lambda), m_0, m_1)$.

Let $S_{\mathcal{TD}}$ be the set of identities queried by $\mathcal{A}$. Here, $O(\cdot)$ is an oracle that has msk hardwired, takes as input an identity $\mathsf{id} \in \{0,1\}^\kappa$ and outputs $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$.

Figure 8.3: Experiment Expt-TT-emb

Based on the above experiment, we now define the following (probabilistic) events and the corresponding probabilities (which are a functions of λ, parameterized by $\mathcal{A}, \epsilon, p$):

- $\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$

  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon,p}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder} \wedge p(\lambda) \geq |S_{\mathfrak{TD}}|]$.

- $\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathfrak{TD}}$

  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

- $\mathsf{Fal\text{-}Tr}$: $T \not\subseteq S_{\mathfrak{TD}}$

  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

A traitor tracing scheme $\mathfrak{T}$ is said to be secure if for every PPT adversary $\mathcal{A}$, polynomials $q(\cdot)$, $p(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists negligible functions $\mathrm{negl}_1(\cdot)$, $\mathrm{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, the following holds

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \mathrm{negl}_1(\lambda), \quad \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon,p}(\lambda) - \mathrm{negl}_2(\lambda).$$

Remark 8.1.2. Note that unlike Definitions 8.1.2 and 8.1.3, here the trace algorithm takes an additional parameter Q_{bd}. In the correct trace definition, we require that as long as the tracing algorithm uses a bound greater than the number of keys queried, the tracing algorithm must identify at least one traitor. However, the false trace guarantee should hold for all polynomially bounded Q_{bd} values. In particular, even if the number of keys queried is

more than the bound used in tracing, the trace algorithm must not output an identity that was not queried. We can show that this definition implies the 'standard' tracing definition where the trace algorithm does not take this bound as input. One simply needs to run this bounded-version of trace with increasing powers of two until the trace algorithm outputs at least one traitor.

The tracing algorithm in [104] also requires a bound q. However, in their definition, the false trace and correct trace events are defined only when the tracing bound is equal to the number of keys queried. As a result, it is not clear if this definition can be used to achieve the 'standard' tracing definition.

8.2 Indexed EITT to Bounded EITT

In this section, we will show how to build a (bounded keys, public/private tracing)-embedded identity tracing scheme from a (indexed keygen, public/private)-embedded identity tracing scheme.

8.2.1 Construction

Let $\mathsf{TT\text{-}ind} = (\mathsf{Ind.Setup}, \mathsf{Ind.Enc}, \mathsf{Ind.KeyGen}, \mathsf{Ind.Dec}, \mathsf{Ind.Trace})$ be a (indexed keygen, public/private)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$, identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$, and with ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{T\text{-}t}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)-efficiency, and let $\mathcal{S} = (\mathsf{SS.Setup}, \mathsf{SS.Sign}, \mathsf{SS.Verify})$ be a signature scheme with message space $\{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$ and signature space $\left\{\{0,1\}^{\ell_{\mathsf{ss}}(\lambda)}\right\}_{\lambda\in\mathbb{N}}$. We use $\mathsf{TT\text{-}ind}$ to build a non-indexed traitor tracing scheme $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ as follows.

$\mathsf{Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}}) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the $\mathsf{TT\text{-}ind}$ setup algorithm λ times with index value $n_{\mathrm{indx}} = 2 \cdot n_{\mathrm{bd}}^2$ and identity index $\kappa + \ell_{\mathsf{ss}}(\lambda)$ as follows:

$$\forall i \in [\lambda], \quad (\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{Ind.Setup}(1^\lambda, 1^{\kappa+\ell_{\mathsf{ss}}(\lambda)}, n_{\mathrm{indx}} = 2 \cdot n_{\mathrm{bd}}^2).$$

It also chooses a signing/verification keys $(\mathsf{ss.sk}, \mathsf{ss.vk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^\kappa)$. It then sets the master secret and public keys as $\mathsf{msk} = \Big((\mathsf{msk}_i)_{i\in[\lambda]}, \mathsf{ss.sk}, \mathsf{ss.vk}\Big)$, $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$ and $\mathsf{key} = \mathsf{ss.vk}, (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}$. Let $\mathsf{msk} = \Big((\mathsf{msk}_i)_{i\in[\lambda]}, \mathsf{ss.sk}, \mathsf{ss.vk}\Big)$. The key generation algorithm chooses λ uniformly random indices $j_i \leftarrow [2 \cdot n_{\mathrm{bd}}^2]$ for $i \in [\lambda]$. Next, it computes a signature on id; that is, it computes $\sigma \leftarrow \mathsf{SS.Sign}(\mathsf{ss.sk}, \mathsf{id})$. Finally, for i^{th} subsystem, it computes indexed keys for index $\{j_i\}$ and identity (id, σ). That is, for $i \in [\lambda]$, it computes $\mathsf{sk}_i \leftarrow \mathsf{Ind.KeyGen}(\mathsf{msk}_i, (\mathsf{id}, \sigma), j_i)$, and outputs the secret key sk as $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\lambda]}$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. Let $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$. The encryption algorithm first chooses $\lambda - 1$ random messages as $r_i \leftarrow \mathcal{M}$ for $i \in [\lambda - 1]$. Next, it sets $r_\lambda = m \oplus \left(\bigoplus_{i=1}^{\lambda-1} r_i\right)$. It then encrypts messages r_i under key pk_i as follows:

$$\forall i \in [\lambda], \quad \mathsf{ct}_i \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_i, r_i).$$

Finally, it outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. Let $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\lambda]}$, and $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$. The decryption algorithm runs the TT-ind decryption on each secret key-ciphertext pair as $z_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_i)$ for $i \in [\lambda]$.

If $z_i = \perp$ for any $i \in [\lambda]$, then it outputs $z = \perp$, otherwise it outputs $z = \bigoplus_{i=1}^{\lambda} z_i$ as the message.

$\mathsf{isGoodDecoder}^D((\mathsf{pk}_i)_{i\in[\lambda]}, 1^y, m_0, m_1, r, i)$

Input: Keys $(\mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: Yes/No.

1. Set $\mathsf{count} = 0$. (Let $\epsilon = 1/y$.)
2. For $j = 1$ to $\lambda \cdot y$:
   - Choose $\lambda - 1$ messages r_k randomly for $k \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{k\in[\lambda]\setminus\{i\}} r_k = r$. (That is, bit-wise parity of the messages chosen matches the message r.)
   - Choose random bit $b \leftarrow \{0,1\}$, and compute ciphertexts as $\mathsf{ct}_k \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_k, r_k)$ for $k \in [\lambda] \setminus \{i\}$, and $\mathsf{ct}_i \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_i, r \oplus m_b)$.
   - Query ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda)$ to the oracle D. Let b' denote the oracle's response.
   - If $b = b'$, set $\mathsf{count} = \mathsf{count} + 1$.
3. If $\mathsf{count}/(\lambda \cdot y) \geq 1/2 + \epsilon/3$, then output 'Yes'. Otherwise output 'No'.

Figure 8.4: Routine isGoodDecoder

$\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1) \to T$. Let $\mathsf{key} = \left((\mathsf{msk}_i, \mathsf{pk}_i)_{i\in[\lambda]}, \mathsf{ss.vk}\right)$ and $\epsilon = 1/y$. First we define a supplementary algorithms isGoodDecoder (in Figure 8.4) and SubTrace (in Figure 8.5) that both get oracle access to the decoder D and take as input all λ public-secret key pairs $(\mathsf{msk}_i, \mathsf{pk}_i)_i$, parameter y, messages m_0, m_1, r and an index $i \in [\lambda]$. The tracing algorithm executes

$\mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$

Input: Keys $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: $T \subseteq \{0,1\}^{\kappa}$.

1. It runs the $\mathsf{TT\text{-}ind}$ tracing algorithm on inputs — keys $\mathsf{key}_i, \mathsf{pk}_i$, parameter $4y$, messages $m_0 \oplus r, m_1 \oplus r$ — and with oracle access to oracle $\widetilde{D}$ which we define next.

2. On each query ct to oracle $\widetilde{D}$ by $\mathsf{Ind.Trace}$, the sub-tracing algorithm first chooses $\lambda - 1$ messages r_j randomly for $j \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r$. (That is, bit-wise parity of the messages chosen matches the message r.) It encrypts these messages as $\mathsf{ct}_j \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_j, r_j)$, and then sends the ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda)$ to the oracle D as its query. And it forwards the oracle D's response to the $\mathsf{Ind.Trace}$ algorithm.

3. Finally, the $\mathsf{Ind.Trace}$ algorithm outputs a set T. The sub-tracing algorithm outputs the same set T as its output.

$$\text{In short,} \quad \mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$$
$$= \mathsf{Ind.Trace}^{\widetilde{D}}((\mathsf{msk}_i, \mathsf{pk}_i), 1^{4y}, m_0 \oplus r, m_1 \oplus r),$$
$$\widetilde{D}(\mathsf{ct}) = D(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda),$$
$$\text{where} \quad \forall j \in [\lambda] \setminus \{i\}\,,\ r_j \leftarrow \mathcal{M} \text{ such that } \bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r,$$
$$\forall j \in [\lambda] \setminus \{i\}\,,\ \mathsf{ct}_j \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_j, r_j)$$

Figure 8.5: Routine $\mathsf{SubTrace}$

the following procedure using $\mathsf{isGoodDecoder}$ and $\mathsf{SubTrace}$ routines as follows:

1. Set $i = 1$.

2. Set flag = 'No'. For $j = 1$ to $\lambda \cdot y$:

   - Choose a random message $r \leftarrow \mathcal{M}$.

- Run $\mathsf{isGoodDecoder}$ as $\mathsf{flag} \leftarrow \mathsf{isGoodDecoder}^{D}(\mathsf{key}, 1^{y}, m_0, m_1, r, i)$.
- If flag = 'Yes', break. Else, continue.

3. If flag = 'Yes', run $\mathsf{SubTrace}$ as $T \leftarrow \mathsf{SubTrace}^{D}(\mathsf{key}, 1^{y}, m_0, m_1, r, i)$. Else, set $T = \emptyset$.
4. If $T = \emptyset$ and $i < \lambda$, set $i = i + 1$ and go to step 2. Otherwise, output T.

Set $T^{\text{final}} = \emptyset$. For each $(\mathsf{id}, \sigma) = \tilde{\mathsf{id}} \in T$, if $\mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1$, the tracing algorithm adds id to T^{final}. Concretely,

$$T^{\text{final}} = \{\mathsf{id} \ : \ \exists\ \sigma \text{ such that } (\mathsf{id}, \sigma) \in T \text{ and } \mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1\}\,.$$

Finally, it outputs the set T^{final} as the set of traitors.

8.2.2 Correctness and Security

Next, we prove the following.

Theorem 8.2.1. *If* $\mathsf{TT\text{-}ind} = (\mathsf{Ind.Setup}, \mathsf{Ind.Enc}, \mathsf{Ind.KeyGen}, \mathsf{Ind.Dec}, \mathsf{Ind.Trace})$ *is a secure (indexed keygen, public/private)-embedded identity tracing scheme (as per Definitions 8.1.1 and 4.1.2) with* $(\mathsf{T\text{-}s}, \mathsf{T\text{-}e}, \mathsf{T\text{-}k}, \mathsf{T\text{-}d}, \mathsf{T\text{-}t}, \mathsf{S\text{-}c}, \mathsf{S\text{-}k})$*-efficiency, and* $\mathcal{S} = (\mathsf{SS.Setup}, \mathsf{SS.Sign}, \mathsf{SS.Verify})$ *is a secure signature scheme (as per Definition 3.4.2), then the scheme* $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ *(described in Section 8.2.1) is a secure (bounded keys, public/private tracing)-embedded identity tracing scheme (as per Definitions 8.1.1 and 8.1.3) with* $(\mathsf{T\text{-}s}', \mathsf{T\text{-}e}', \mathsf{T\text{-}k}', \mathsf{T\text{-}d}', \mathsf{T\text{-}t}', \mathsf{S\text{-}c}', \mathsf{S\text{-}k}')$*-efficiency, where the efficiency measures are related as follows:*

- $\mathsf{T\text{-}s}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}s}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}k}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}k}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}e}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}e}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}d}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{T\text{-}d}(\lambda, \kappa, 2 \cdot n^2) + \mathsf{poly}(\lambda, \kappa, \log n)$,
- $\mathsf{T\text{-}t}'(\lambda, \kappa, n, y) = \lambda \cdot \mathsf{T\text{-}t}(\lambda, \kappa, 2 \cdot n^2, 4y) + \lambda^3 \cdot y^2$,
- $\mathsf{S\text{-}c}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{S\text{-}c}(\lambda, \kappa, 2 \cdot n^2)$,
- $\mathsf{S\text{-}k}'(\lambda, \kappa, n) = \lambda \cdot \mathsf{S\text{-}k}(\lambda, \kappa, 2 \cdot n^2)$.

Proof. **Correctness**: Fix any security parameter λ, public key $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$, master secret key $\mathsf{msk} = \left((\mathsf{msk}_i)_{i\in[\lambda]}, (\mathsf{ss.sk}, \mathsf{ss.vk})\right)$, tracing key $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)$, message $m \in \mathcal{M}$ and identity id. The encryption algorithm chooses $\{r_i\}_{i\in[\lambda]}$ such that $\bigoplus_i r_i = m$, computes $\mathsf{ct}_i \leftarrow \mathsf{Ind.Enc}(\mathsf{pk}_i, r_i)$ and sets $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$. The key generation algorithm first chooses λ indices $\{j_i\}_{i\in[\lambda]}$, computes a signature σ on id, and outputs λ keys $(\mathsf{sk}_i)_{i\in[\lambda]}$, where sk_i is a key for identity (id, σ) and index j_i. From the correctness of the underlying scheme $\mathsf{TT\text{-}ind}$, it follows that decryption of ct_i using sk_i outputs r_i. As a result, the decryption of ct using sk outputs message m.

IND-CPA security: $\mathsf{IND\text{-}CPA}$ security of our scheme follows directly from the $\mathsf{IND\text{-}CPA}$ security of $\mathsf{TT\text{-}ind}$. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks the $\mathsf{IND\text{-}CPA}$ security of our scheme with advantage ϵ. We can use $\mathcal{A}$ to break the $\mathsf{IND\text{-}CPA}$ security of $\mathsf{TT\text{-}ind}$ with advantage ϵ. The reduction algorithm first receives pk_1 from the $\mathsf{TT\text{-}ind}$ challenger. It chooses $(\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{Ind.Setup}(1^\lambda, 1^\kappa, n_{\mathrm{indx}})$ for each $i \in \{2, \ldots, \lambda\}$, and sends

$(\mathsf{pk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. (If $\mathsf{TT\text{-}ind}$ is a public tracing scheme, then the reduction algorithm also receives a tracing key key_1 from the challenger, and it sends the tracing key $(\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$).

Next, $\mathcal{A}$ sends two messages m_0, m_1 to $\mathcal{B}$. The reduction algorithm chooses $r_2, \ldots, r_n \leftarrow \mathcal{M}$, sets $m'_b = m_b \oplus \left(\bigoplus_{i>1} r_i\right)$ and sends (m'_0, m'_1) to the challenger. The challenger sends ct_1 to $\mathcal{B}$. The reduction algorithm computes encryptions of $r_2, \ldots, r_\lambda$, and sends $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. The adversary sends its guess, which the reduction algorithm forwards to the challenger.

Clearly, if $\mathcal{A}$ has advantage ϵ, then so does $\mathcal{B}$.

Efficiency: It is easy to verify that $\mathsf{T\text{-}s'}$, $\mathsf{T\text{-}k'}$, $\mathsf{T\text{-}e'}$, $\mathsf{T\text{-}d'}$, $\mathsf{T\text{-}t'}$, $\mathsf{S\text{-}c'}$ and $\mathsf{S\text{-}k'}$ satisfy the efficiency measures.

Correct Trace and False Trace guarantees: We will now show that our scheme satisfies Definition 8.1.3.

False Trace: We will first show that the false trace probability of our scheme is negligible in the security parameter. Recall, the false trace probability is defined as follows: the adversary receives the public key (and the tracing key in a public tracing algorithm). Next, it queries for a set of keys $S_{\mathcal{ID}}$ corresponding to a set of identities (note that in the false trace experiment, the adversary can query for an unbounded number of keys). Finally, it outputs a decoding box D and two messages m_0, m_1. The tracing algorithm uses D and m_0, m_1 to output a set of traitors T^{final}. In particular, the tracing algorithm uses the $\mathsf{SubTrace}$ algorithm to find a polynomial sized set $T = \{(\mathsf{id}_j, \sigma_j)\}_j$,

and puts an identity id in T^{final} if $(\mathsf{id}, \sigma) \in T$ and $\mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1$. The false trace probability is the probability that $T^{\text{final}} \not\subseteq S_{\mathcal{ID}}$.

Suppose there exists a PPT adversary $\mathcal{A}$ such that the false trace probability is ϵ. We can use $\mathcal{A}$ to break the signature scheme's security. The reduction algorithm $\mathcal{B}$ receives the verification key $\mathsf{ss.vk}$ from the challenger. It chooses λ public/master secret/tracing keys $\{(\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i)\}$ for the underlying traitor tracing system $\mathsf{TT\text{-}ind}$, and sends $(\mathsf{pk}_i)_{i\in[\lambda]}$ to the adversary (if $\mathsf{TT\text{-}ind}$ is public tracing, then it also sends $\left((\mathsf{key}_i)_{i\in[\lambda]}, \mathsf{ss.vk}\right)$ as the tracing key). Next, the adversary gets access to a key generation oracle. For every query id, the reduction algorithm sends id to the challenger, and receives signature σ. It then chooses λ uniformly random indices $\{j_i\}_{i\in[\lambda]}$ and generates indexed keys $\mathsf{sk}_j \leftarrow \mathsf{Ind.KeyGen}(\mathsf{msk}_i, (\mathsf{id}, \sigma), j_i)$ for each $i \in [\lambda]$. It sends $(\mathsf{sk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. Finally, after polynomially many secret key queries, the adversary outputs a decoding box D and messages m_0, m_1. The reduction algorithm first checks if it is a good decoder (that is, steps 1 and 2 of the tracing algorithm). Next, if it is a good decoder, it runs $\mathsf{SubTrace}$ to obtain a set $T = \{\mathsf{id}_j, \sigma_j\}_j$. If there exists an (id, σ) pair in T such that id was not queried by $\mathcal{A}$ and $\mathsf{SS.Verify}(\mathsf{ss.vk}, \mathsf{id}, \sigma) = 1$, then it sends id to the challenger. Since the false trace event happens with probability ϵ, the reduction algorithm breaks the security of the signature scheme with probability ϵ.

Remark 8.2.1. Note that we do not require the 'false trace' guarantee of the underlying tracing scheme $\mathsf{TT\text{-}ind}$. We only require that the tracing algorithm of the underlying scheme is polynomial time.

Correct Trace: Recall the traitor tracing experiment (Figure 8.2): the challenger sends a public key pk, then the adversary queries for a set of secret keys (at most n_{bd} keys) corresponding to identities. Finally, the adversary outputs a decoding box D. We need to show that if the decoding box if ϵ good, then the tracing algorithm can trace at least one identity for which the adversary queried a secret key. More formally, the correct trace guarantee is captured by the following events/probabilities:

$\mathsf{Good\text{-}Decoder}$: $\Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)] \geq 1/2 + \epsilon(\lambda)$

$\mathsf{Admissible\text{-}Adv}$: $\mathcal{A}$ makes at most n_{bd} key queries

$\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}]$.

$\mathsf{Cor\text{-}Tr}$: $T \neq \emptyset \wedge T \subseteq S_{\mathfrak{ID}}$, $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$.

We require that $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}_2(\lambda)$.

We will define some more events and their probabilities, which will be useful for our proof.

- Tracing without correctness: Event Tr, which is similar to $\mathsf{Cor\text{-}Tr}$, except that we do not require $T \subseteq S_{\mathfrak{ID}}$. More formally, let Tr : $T \neq \emptyset$, and the corresponding probability $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Tr}]$.

- Position with distinct indices for each key: Recall in our construction, each key for identity id consists of λ different 'indexed' keys, where the λ indices are chosen uniformly at random from $[2n_{\mathrm{bd}}^2]$. Let $\mathsf{Dist\text{-}Indx}$ be

the event that there exists $i \in [\lambda]$ such that the i^{th} index of each key is distinct.

- $\mathsf{Dist\text{-}Indx}_i$: Position i is the first position such that the i^{th} index of each key is distinct. Hence, by definition, $\mathsf{Dist\text{-}Indx} = \bigcup_i \mathsf{Dist\text{-}Indx}_i$.

- Tracing without correctness in i^{th} iteration: Let T_i denote the set of (identity, signatute) pairs traced in the i^{th} iteration. The event Tr_i happens if T_i is non-empty.

- Tracing with same signature as that received in key: Let T_i denote the set of (identity, signature) pairs that are traced in the i^{th} iteration. The event $\mathsf{Cor\text{-}Tr\text{-}Sig}_i$ happens if for all $(\mathsf{id}, \sigma) \in T_i$, id was queried during the key query phase, and the key generation oracle output $\mathsf{sk} \leftarrow \mathsf{Ind.KeyGen}(\mathsf{msk}_i, (\mathsf{id}, \sigma), j)$ for some index j.

- The flag is set to 'Yes' in the i^{th} iteration: The event $\mathsf{Found\text{-}Good\text{-}r}_i$ happens if in the i^{th} iteration, the flag is set to 'Yes'.

- Good decoder $\widetilde{D}$ during the $\mathsf{SubTrace}$ routine execution in i^{th} iteration: We say that event $\mathsf{Good\text{-}\widetilde{D}}_i$ happens if in the i^{th} iteration, the execution reaches step 3 (that is, it found a 'good' r in the i^{th} iteration), and the decoder $\widetilde{D}$ constructed is an $\epsilon/4$ good decoder for distinguishing messages $m_0 \oplus r$ and $m_1 \oplus r$. Note that if no good r is found in step 3, then we say that $\mathsf{Good\text{-}\widetilde{D}}_i$ did not happen.

With these events defined, we will now show that $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$. The proof will proceed via a series of inequalities as shown below.

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \tag{8.1}$$

$$\geq \Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}_1(\lambda) \tag{8.2}$$

$$\geq \Pr\left[\mathsf{Tr} \wedge \mathsf{Dist\text{-}Indx}\right] - \mathrm{negl}_1(\lambda) \tag{8.3}$$

$$= \sum_i \Pr\left[\mathsf{Tr} \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathrm{negl}_1(\lambda) \tag{8.4}$$

$$\geq \sum_i \Pr\left[\mathsf{Cor\text{-}Tr\text{-}Sig}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathrm{negl}_1(\lambda) \tag{8.5}$$

$$\geq \sum_i \Pr\left[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathrm{negl}_2(\lambda) \tag{8.6}$$

$$\geq \sum_i \Pr\left[\begin{array}{c}\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder} \\ \wedge\, \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i\end{array}\right] - \mathrm{negl}_2(\lambda) \tag{8.7}$$

$$\geq \sum_i \Pr\left[\begin{array}{c}\mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder} \\ \wedge\, \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i\end{array}\right] - \mathrm{negl}_3(\lambda) \tag{8.8}$$

$$\geq \sum_i \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i\right] - \mathrm{negl}_4(\lambda) \tag{8.9}$$

$$= \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}\right] - \mathrm{negl}_4(\lambda) \tag{8.10}$$

$$\geq \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}\right] - \mathrm{negl}_5(\lambda) \tag{8.11}$$

We will now discuss each of these steps, and why the above inequalities hold. Step 8.1 to 8.2 follows from the false trace guarantee. Using the false trace guarantee, we can show that $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \mathrm{negl}(\lambda)$.

Claim 8.2.1. $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \mathit{negl}(\lambda)$.

Proof. From the definition of the events, it follows that $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda)+$

$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda)$. Next, using the false trace guarantees, we can argue that $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) + \mathrm{negl}(\lambda)$. ■

Step 8.2 to 8.3 follows from definition of $\Pr\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda)$. Step 8.3 to 8.4 follows from the fact that $\mathsf{Dist\text{-}Indx} = \cup_i \mathsf{Dist\text{-}Indx}_i$, and these events are mutually exclusive. Step 8.4 to 8.5 holds because the event $\mathsf{Cor\text{-}Tr\text{-}Sig}_i$ implies Tr (we are assuming our signature scheme is perfectly correct, and hence any signature generated by the signing key is accepted by the verification algorithm). Step 8.5 to 8.6 follows from the correct-trace guarantee of the underlying $\mathsf{TT\text{-}ind}$ scheme. This is formalized in the following claim.

Claim 8.2.2. *Assuming the traitor tracing scheme satisfies the correct-trace guarantee as defined in Definition 4.1.2, there exists a negligible function $negl(\cdot)$ such that for all $i \in [\lambda]$,*

$$\Pr\left[\mathsf{Cor\text{-}Tr\text{-}Sig}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] \geq \Pr\left[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Dist\text{-}Indx}_i\right] - negl(\lambda)$$

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\Pr[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i] - \Pr\left[\mathsf{Cor\text{-}Tr\text{-}Sig}_i \wedge \mathsf{Dist\text{-}Indx}_i\right]$ is non-negligible. We will use $\mathcal{A}$ to build a PPT algorithm $\mathcal{B}$ that breaks the security of $\mathsf{TT\text{-}ind}$. The reduction algorithm first receives the bound n_{bd} (in unary) from the adversary. It sets $n_{\mathrm{indx}} = 2 \cdot n_{\mathrm{bd}}^2$, sends n_{indx} to the challenger (in unary) and receives the public key pk_i (and key_i if $\mathsf{TT\text{-}ind}$ is a public tracing scheme). For each $k \neq i$, it chooses $(\mathsf{pk}_k, \mathsf{msk}_k, \mathsf{key}_k) \leftarrow \mathsf{Ind.Setup}(1^\lambda, 1^{\kappa+\ell_{\mathsf{ss}}(\lambda)}, n_{\mathrm{indx}})$, chooses a signing/verification key, and sends the public key to $\mathcal{A}$ (and the tracing key if

applicable). Next, the adversary queries for secret keys. For each query id, the reduction algorithm chooses λ indices $j_{\mathsf{id},1}, \ldots, j_{\mathsf{id},\lambda}$. It computes a signature σ on id and queries for a TT-ind key sk_i corresponding to (id, σ) for index $j_{\mathsf{id},i}$. For $k \neq i$, it computes a sk_k using msk_k. Finally, it sends $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\lambda)$ to $\mathcal{A}$. After at most n_{bd} key queries, the adversary submits a decoding box D and messages m_0, m_1. The reduction algorithm first checks if, for all key queries, the i^{th} position's indices are distinct. If not, it outputs a dummy decoding box and quits. Next, it runs $\mathsf{isGoodDecoder}((\mathsf{pk}_i)_{i\in[\lambda]}, 1^y, m_0, m_1, r)$ for uniformly random and independently chosen r, until it finds an r s.t. isGoodDecoder outputs 'Yes'. If no such r is found after $\lambda \cdot y$ iterations, it quits (and outputs a dummy decoding box). It constructs $\widetilde{D}$ using D (as described in SubTrace) and sets the distinguishing messages to be $\widetilde{m_0} = m_0 + r$, $\widetilde{m_1} = m_1 + r$. It sends $\widetilde{D}, \widetilde{m_0}, \widetilde{m_1}$ to the challenger.

First, let us compute the probability that the reduction algorithm outputs a $1/4y$ good decoding box for $\widetilde{m_0}$, $\widetilde{m_1}$. This happens if all the following events happen:

- All the position i indices of the keys are distinct.
- The decoder $\widetilde{D}$ is a $1/4y$ good decoder for $\widetilde{m_0}, \widetilde{m_1}$. This happens only if isGoodDecoder outputs 'Yes' for one of the randomly chosen r values.

The probability that $\mathcal{B}$ outputs a good decoding box is $\Pr[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}_i]$. Using the security of TT-ind, $\Pr[\mathsf{Cor\text{-}Tr\text{-}Sig}_i \wedge \mathsf{Dist\text{-}Indx}_i] \geq$

$\Pr[\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i \wedge \mathsf{Dist\text{-}Indx}_i] - \mathrm{negl}(\lambda)$. Here, note that the set of (identity, signature) pairs output by the tracing algorithm is a subset of the (identity, signature) queries made by the reduction algorithm. This concludes our proof. ■

Step 8.6 to 8.7 follows directly from the definition of the events. Next, we will show that $\mathsf{Found\text{-}Good\text{-}r}_i$ and $\overline{\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i}$ happens with negligible probability. This justifies the step 8.7 to 8.8 inequality.

Claim 8.2.3. *There exists a negligible function $negl(\cdot)$ such that for any index i,* $\Pr\left[\mathsf{Found\text{-}Good\text{-}r}_i \wedge \overline{\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i}\right] \leq negl(\lambda)$.

Proof. Using Chernoff bounds, we can argue that $\Pr\left[\mathsf{Found\text{-}Good\text{-}r}_i \mid \overline{\mathsf{Good\text{-}}\widetilde{\mathsf{D}}_i}\right] < \mathrm{negl}(\lambda)$, and hence the claim follows. ■

We will now show that for every i, $\mathsf{Good\text{-}Decoder}$ and $\overline{\mathsf{Found\text{-}Good\text{-}r}_i}$ happen with negligible probability. Hence, we can justify the step 8.8 to 8.9 transition.

Claim 8.2.4. *There exists a negligible function $negl(\cdot)$ such that for any index i,* $\Pr\left[\overline{\mathsf{Found\text{-}Good\text{-}r}_i} \wedge \mathsf{Good\text{-}Decoder}\right] \leq negl(\lambda)$.

Proof. As in the previous proof, this also follows via a simple application of Chernoff bounds, since the term $\Pr\left[\overline{\mathsf{Found\text{-}Good\text{-}r}_i} \mid \mathsf{Good\text{-}Decoder}\right]$ can be bounded by $\mathrm{negl}(\lambda)$.

■

Next, since $\cup_i \mathsf{Dist\text{-}Indx}_i = \mathsf{Dist\text{-}Indx}$, and all the $\mathsf{Dist\text{-}Indx}_i$ are mutually exclusive, step 8.9 to 8.10 follows. Finally, using the following claim, we can argue the step 8.10 to 8.11 inequality.

Claim 8.2.5. *There exist a negligible function* $negl(\cdot)$ *such that* $\Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}] \geq \Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}] - negl(\lambda)$.

Proof. Consider the following experiment: choose $x_{i,j} \leftarrow [2 \cdot n_{\text{bd}}^2]$ for each $i \in [\lambda]$, $j \in [n_{\text{bd}}]$ independently. Let $Y_i = 1$ if the set $\{x_{i,j}\}_{j \in [\lambda]}$ consists of n_{bd} distinct entries, else $Y_i = 0$. Using a simple union bound, it follows that $\Pr[Y_i = 0] \leq 1/4$, and since all Y_i are independent, $\Pr[\ \forall i,\ Y_i = 0] \leq 1/4^{\lambda}$.

From the above experiment, it follows that $\Pr[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \overline{\mathsf{Dist\text{-}Indx}}] \leq \text{negl}(\lambda)$. As a result,

$$
\begin{aligned}
&\Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv}\right] \\
&= \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}\right] \\
&\quad + \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \overline{\mathsf{Dist\text{-}Indx}}\right] \\
&\leq \Pr\left[\mathsf{Good\text{-}Decoder} \wedge \mathsf{Admissible\text{-}Adv} \wedge \mathsf{Dist\text{-}Indx}\right] + \text{negl}(\lambda)
\end{aligned}
$$

■

■

8.3 Bounded EITT to Unbounded EITT

In this section, we provide an efficient generic transformation that removes the bound set on the number of users/keys that an adversary is allowed to corrupt during setup.

8.3.1 Construction

Let $\mathsf{TT\text{-}bd} = (\mathsf{BD.Setup}, \mathsf{BD.KeyGen}, \mathsf{BD.Enc}, \mathsf{BD.Dec}, \mathsf{BD.Trace})$ be a (bounded keys, public/private tracing)-embedded identity tracing scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$, identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa\in\mathbb{N}}$, and with ($\mathsf{T\text{-}setup}$, $\mathsf{T\text{-}enc}$, $\mathsf{T\text{-}key}$)-efficiency. We use $\mathsf{TT\text{-}bd}$ to build a traitor tracing $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ with unbounded number of users as follows. (Here we provide a transformation for TT schemes with secret key tracing, but the construction can be easily extended to work in the public tracing setting as well.)

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the $\mathsf{TT\text{-}bd}$ setup algorithm λ times with increasing values of the user bound n_{bd} as follows:

$$\forall i \in [\lambda], \quad (\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{BD.Setup}(1^\lambda, 1^\kappa, n_{\mathrm{bd}} = 2^i).$$

It then sets the master secret and public keys as an λ-tuple of all these keys, i.e. $\mathsf{msk} = (\mathsf{msk}_i)_{i\in[\lambda]}$, $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$ and $\mathsf{key} = (\mathsf{pk}_i, \mathsf{key}_i)_{i\in[\lambda]}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}$. Let $\mathsf{msk} = (\mathsf{msk}_i)_{i\in[\lambda]}$. The key generation algorithm runs the $\mathsf{TT\text{-}bd}$ key generation algorithm with all λ keys independently

as $\mathsf{sk}_i \leftarrow \mathsf{BD.KeyGen}(\mathsf{msk}_i, \mathsf{id})$ for $i \in [\lambda]$. It outputs the secret key sk as $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\lambda]}$.

$\mathsf{Enc}(\mathsf{pk}, m) \rightarrow \mathsf{ct}$. Let $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$. The encryption algorithm first chooses $\lambda - 1$ random messages as $r_i \leftarrow \mathcal{M}$ for $i \in [\lambda - 1]$. Next, it sets $r_\lambda = m \oplus \left(\bigoplus_{i=1}^{\lambda-1} r_i\right)$. It then encrypts messages r_i under key pk_i as follows:

$$\forall i \in [\lambda], \quad \mathsf{ct}_i \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_i, r_i).$$

Finally, it outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow z$. Let $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\lambda]}$, and $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$. The decryption algorithm runs the $\mathsf{TT\text{-}bd}$ decryption on each secret key-ciphertext pair as $z_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}_i)$ for $i \in [\lambda]$.

If $z_i = \perp$ for any $i \in [\lambda]$, then it outputs $z = \perp$, otherwise it outputs $z = \bigoplus_{i=1}^{\lambda} z_i$ as the message.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, Q_{\mathrm{bd}}, m_0, m_1) \rightarrow T$. Let $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$ and $\epsilon = 1/y$. First we define a supplementary algorithms $\mathsf{isGoodDecoder}$ (in Figure 8.6) and $\mathsf{SubTrace}$ (in Figure 8.7) that both get oracle access to the decoder D and take as input all λ tracing/public key pairs $(\mathsf{key}_i, \mathsf{pk}_i)_i$, parameter y, messages m_0, m_1, r and an index $i \in [\lambda]$. The tracing algorithm executes the following procedure using $\mathsf{isGoodDecoder}$ and $\mathsf{SubTrace}$ routines as follows:

1. Set $i = \lceil \log Q_{\mathrm{bd}} \rceil$.

$\mathsf{isGoodDecoder}^{D}((\mathsf{pk}_i), 1^y, m_0, m_1, r, i)$

Input: Public keys $(\mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: Yes/No.

1. Set $\mathsf{count} = 0$. (Let $\epsilon = 1/y$.)
2. For $j = 1$ to $\lambda \cdot y$:
   - Choose $\lambda - 1$ messages r_k randomly for $k \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{k\in[\lambda]\setminus\{i\}} r_k = r$. (That is, bit-wise parity of the messages chosen matches the message r.)
   - Choose random bit $b \leftarrow \{0,1\}$, and compute ciphertexts as $\mathsf{ct}_k \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_k, r_k)$ for $k \in [\lambda] \setminus \{i\}$, and $\mathsf{ct}_i \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_i, r \oplus m_b)$.
   - Query ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda)$ to the oracle D. Let b' denote the oracle's response.
   - If $b = b'$, set $\mathsf{count} = \mathsf{count} + 1$.
3. If $\mathsf{count}/(\lambda \cdot y) \geq 1/2 + \epsilon/3$, then output 'Yes'. Otherwise output 'No'.

Figure 8.6: Routine isGoodDecoder

2. Set flag = 'No'. For $j = 1$ to $\lambda \cdot y$:
   - Choose a random message $r \leftarrow \mathcal{M}$.
   - Run isGoodDecoder as $\mathsf{flag} \leftarrow \mathsf{isGoodDecoder}^{D}(\left(\mathsf{pk}_j\right), 1^y, m_0, m_1, r, i)$.
   - If flag = 'Yes', break. Else, continue.
3. If flag = 'Yes', run SubTrace as $T \leftarrow \mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$.
   Else, set $T = \emptyset$.
4. Output T.

$\mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$

Input: Keys $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$, Parameter y, Messages m_0, m_1, r, Index $i \in [\lambda]$.
Output: $T \subseteq \{0,1\}^{\kappa}$.

1. It runs the TT-bd tracing algorithm on inputs — keys $\mathsf{key}_i, \mathsf{pk}_i$, parameter $4y$, messages $m_0 \oplus r, m_1 \oplus r$ — and with oracle access to oracle $\widetilde{D}$ which we define next.

2. On each query ct to oracle $\widetilde{D}$ by BD.Trace, the sub-tracing algorithm first chooses $\lambda - 1$ messages r_j randomly for $j \in [\lambda] \setminus \{i\}$ such that $\bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r$. (That is, bit-wise parity of the messages chosen matches the message r.) It encrypts these messages as $\mathsf{ct}_j \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_j, r_j)$, and then sends the ciphertext $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda)$ to the oracle D as its query. And it forwards the oracle D's response to the BD.Trace algorithm.

3. Finally, the BD.Trace algorithm outputs a set T. The sub-tracing algorithm outputs the same set T as its output.

$$\text{In short,} \quad \mathsf{SubTrace}^{D}(\mathsf{key}, 1^y, m_0, m_1, r, i)$$
$$= \mathsf{BD.Trace}^{\widetilde{D}}((\mathsf{key}_i, \mathsf{pk}_i), 1^{4y}, m_0 \oplus r, m_1 \oplus r),$$
$$\widetilde{D}(\mathsf{ct}) = D(\mathsf{ct}_1, \ldots, \mathsf{ct}_{i-1}, \mathsf{ct}, \mathsf{ct}_{i+1}, \ldots, \mathsf{ct}_\lambda),$$
$$\text{where} \quad \forall j \in [\lambda] \setminus \{i\}\,,\ r_j \leftarrow \mathcal{M} \text{ such that } \bigoplus_{j\in[\lambda]\setminus\{i\}} r_i = r,$$
$$\forall j \in [\lambda] \setminus \{i\}\,,\ \mathsf{ct}_j \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_j, r_j)$$

Figure 8.7: Routine SubTrace

8.3.2 Correctness and Security

Next, we prove the following.

Theorem 8.3.1. *If* TT-bd = (BD.Setup, BD.KeyGen, BD.Enc, BD.Dec, BD.Trace) *is a secure (bounded keys, public/private tracing)-embedded identity tracing scheme (as per Definitions 8.1.3 and 8.1.1) with (*T-s, T-e, T-k, T-d, T-t, S-c,

S-k)*-efficiency, then the scheme* TT = (Setup, KeyGen, Enc, Dec, Trace) *(described in Section 8.3.1) is a secure (unbounded keys, public/private tracing)-embedded identity tracing scheme (as per Definitions 8.1.4 and 8.1.1) with (*T-s′, T-e′, T-k′, T-d′, T-t′, S-c′, S-k′*)-efficiency, where the efficiency measures are related as follows:*

- $\mathsf{T\text{-}s}'(\lambda,\kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}s}(\lambda,\kappa,2^i)$,
- $\mathsf{T\text{-}k}'(\lambda,\kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}k}(\lambda,\kappa,2^i)$,
- $\mathsf{T\text{-}e}'(\lambda,\kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}e}(\lambda,\kappa,2^i) + \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}d}'(\lambda,\kappa) = \sum_{i=1}^{\lambda} \mathsf{T\text{-}d}(\lambda,\kappa,2^i) + \mathsf{poly}(\lambda)$,
- $\mathsf{T\text{-}t}'(\lambda,\kappa,y,Q_{\mathrm{bd}}) = \mathsf{T\text{-}t}(\lambda,\kappa,2^{\lceil \log Q_{\mathrm{bd}} \rceil},4y) + \lambda^2 \cdot y^2$,
- $\mathsf{S\text{-}c}'(\lambda,\kappa) = \sum_{i=1}^{\lambda} \mathsf{S\text{-}c}(\lambda,\kappa,2^i)$,
- $\mathsf{S\text{-}k}'(\lambda,\kappa) = \sum_{i=1}^{\lambda} \mathsf{S\text{-}k}(\lambda,\kappa,2^i)$.

Proof. **Correctness**: Fix any security parameter λ, public key $\mathsf{pk} = (\mathsf{pk}_i)_{i\in[\lambda]}$, master secret key $\mathsf{msk} = \left((\mathsf{msk}_i)_{i\in[\lambda]}\right)_{i\in[\lambda]}$, tracing key $\mathsf{key} = (\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$, message $m \in \mathcal{M}$ and identity id. The encryption algorithm chooses $\{r_i\}_{i\in[\lambda]}$ such that $\bigoplus_i r_i = m$, computes $\mathsf{ct}_i \leftarrow \mathsf{BD.Enc}(\mathsf{pk}_i, r_i)$ and sets $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$. The key generation algorithm outputs λ keys $(\mathsf{sk}_i)_{i\in[\lambda]}$, where sk_i is a key for identity id computed using msk_i. From the correctness of the underlying scheme TT-bd, it follows that decryption of ct_i using sk_i outputs r_i. As a result, the decryption of ct using sk outputs message m.

IND-CPA security: IND-CPA security of our scheme follows directly from the IND-CPA security of TT-bd. Suppose there exists a PPT adversary $\mathcal{A}$ that

breaks the IND-CPA security of our scheme with advantage ϵ. We can use $\mathcal{A}$ to break the IND-CPA security of TT-bd with advantage ϵ. The reduction algorithm first sends λ and bound $= 2$ to the challenger, and receives pk_1 from the TT-ind challenger. It chooses $(\mathsf{msk}_i, \mathsf{pk}_i, \mathsf{key}_i) \leftarrow \mathsf{BD.Setup}(1^\lambda, 1^\kappa, 2^i)$ for each $i \in \{2, \ldots, \lambda\}$, and sends $(\mathsf{pk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. (If TT-bd is a public tracing scheme, then the reduction algorithm also receives a tracing key key_1 from the challenger, and it sends the tracing key $(\mathsf{key}_i, \mathsf{pk}_i)_{i\in[\lambda]}$ to $\mathcal{A}$).

Next, $\mathcal{A}$ sends two messages m_0, m_1 to $\mathcal{B}$. The reduction algorithm chooses $r_2, \ldots, r_n \leftarrow \mathcal{M}$, sets $m'_b = m_b \oplus \left(\bigoplus_{i>1} r_i\right)$ and sends (m'_0, m'_1) to the challenger. The challenger sends ct_1 to $\mathcal{B}$. The reduction algorithm computes encryptions of $r_2, \ldots, r_\lambda$, and sends $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[\lambda]}$ to $\mathcal{A}$. The adversary sends its guess, which the reduction algorithm forwards to the challenger.

Clearly, if $\mathcal{A}$ has advantage ϵ, then so does $\mathcal{B}$.

Correct Trace and False Trace guarantees: We will show that our scheme satisfies Definition 8.1.4.

False Trace: First, let us consider the false trace probability. False trace happens if the sub-tracing algorithm outputs a non-empty set T such that $T \not\subseteq S_{\mathcal{ID}}$, where $S_{\mathcal{ID}}$ is the set of keys queried by the adversary. We will show that if there exists an adversary $\mathcal{A}$, polynomial p and non-negligible functions ϵ, η such that $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \geq \eta(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the false-trace guarantee of TT-bd.

The reduction algorithm $\mathcal{B}$ first sends $p(\lambda)$ (in unary) to the TT-bd

challenger, and let $i = \lceil p(\lambda) \rceil$. It receives a public key pk_i (and a tracing key if it is a public tracing scheme). It chooses $(\mathsf{msk}_j, \mathsf{pk}_j, \mathsf{key}_j)$ for all $j \neq i$, and sends $\left(\mathsf{pk}_j\right)$ to $\mathcal{A}$ (and $\left(\mathsf{key}_j\right)$ for a public tracing scheme). Next, the adversary requests for keys. For each query id, the reduction algorithm computes $\mathsf{sk}_{\mathsf{id},j}$ using msk_j if $j \neq i$. It sends id to the challenger, and receives $\mathsf{sk}_{\mathsf{id},i}$. It sends $\mathsf{sk}_{\mathsf{id}} = (\mathsf{sk}_{\mathsf{id},j})$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a decoding box D and messages m_0, m_1. The reduction algorithm first runs $\mathsf{isGoodDecoder}(\left\{\mathsf{pk}_j\right\}, 1^y, m_0, m_1, r, i)$ for $\lambda \cdot y$ choices of r until it finds an r s.t. $\mathsf{isGoodDecoder}$ outputs 'Yes'. The reduction algorithm outputs $\widetilde{D}$ (as defined in $\mathsf{SubTrace}$) as the decoding box, and $m_0 \oplus r$, $m_1 \oplus r$ as the two messages.

Clearly, if $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \geq \eta(\lambda)$, then $\mathcal{B}$ breaks the false-trace guarantee of $\mathsf{TT\text{-}bd}$.

Correct Trace: As before, our proof will proceed via a sequence of inequalities. The events are defined exactly as in the proof of Theorem 8.2.1. Let $i = \lceil p(\lambda) \rceil$.

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon,p}(\lambda) \tag{8.12}$$

$$= \Pr[\mathsf{Cor\text{-}Tr}_i] \tag{8.13}$$

$$\geq \Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathfrak{I}\mathfrak{D}}|] - \mathrm{negl}_1(\lambda) \tag{8.14}$$

$$\geq \Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathfrak{I}\mathfrak{D}}| \wedge \mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder}] \tag{8.15}$$

$$\geq \Pr[\mathsf{Found\text{-}Good\text{-}r}_i \wedge \mathsf{Good\text{-}Decoder} \wedge p(\lambda) \geq |S_{\mathfrak{I}\mathfrak{D}}|] \tag{8.16}$$

$$\geq \Pr[\mathsf{Good\text{-}Decoder} \wedge p(\lambda) \geq |S_{\mathfrak{I}\mathfrak{D}}|] \tag{8.17}$$

The first equality (Step 8.12 to 8.13) follows from the definition of corret-trace. Next, Step 8.13 to 8.14 follows from the correct-trace guarantee of $\mathsf{TT\text{-}bd}$. This is formalized in the following claim.

Claim 8.3.1. *Assuming* $\mathsf{TT\text{-}bd}$ *satisfies Definition 8.1.3, for any PPT adversary* $\mathcal{A}$*, polynomial* $p(\cdot)$ *and non-negligible function* $\epsilon(\cdot)$*,* $\Pr[\mathsf{Cor\text{-}Tr}_i] \geq \Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathcal{TD}}|] - negl_1(\lambda)$.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, polynomial $p(\cdot)$ and non-negligible functions $\epsilon(\cdot), \eta(\cdot)$ such that $\Pr[\mathsf{Good\text{-}Decoder}_i \wedge p(\lambda) \geq |S_{\mathcal{TD}}|] - \Pr[\mathsf{Cor\text{-}Tr}_i] \geq \eta(\lambda)$. We will use $\mathcal{A}$ to construct a PPT algorithm $\mathcal{B}$ that breaks the correct-trace guarantee of $\mathsf{TT\text{-}bd}$.

Let $i = \lceil \log p(\lambda) \rceil$. The reduction algorithm $\mathcal{B}$ sends $\lambda, p(\lambda)$ (in unary) to the challenger, and receives pk_i (and the tracing key key_i if $\mathsf{TT\text{-}bd}$ is a public tracing scheme). It then chooses $(\mathsf{msk}_j, \mathsf{pk}_j, \mathsf{key}_j) \leftarrow \mathsf{BD.Setup}(1^\lambda, 1^\kappa, 2^j)$ for all $j \neq i$ and sends $\left(\mathsf{pk}_j\right)_{j \in [\lambda]}$. It then receives key queries (at most $p(\lambda)$ key queries). For each key query id, the reduction algorithm generates $\mathsf{sk}_{\mathsf{id},j}$ for $j \neq i$, sends id to the challenger, and receives $\mathsf{sk}_{\mathsf{id},i}$. It sends $\mathsf{sk}_{\mathsf{id}} = (\mathsf{sk}_{\mathsf{id},j})$ to the adversary.

Finally, the adversary sends a decoding box D and messages m_0, m_1. The reduction algorithm runs $\mathsf{isGoodDecoder}$ with different r values until the flag is 'Yes'. Then, it uses that r value to set the decoding box $\widetilde{D}$ (as in $\mathsf{SubTrace}$), messages $m'_b = m_b \oplus r$ and sends $\widetilde{D}, m'_0, m'_1$ to the challenger. ∎

The inequality from 8.14 to 8.15 follows directly from the definition of the events. Next, the justification for transition from Step 8.15 to 8.16 and Step 8.16 to 8.17 is similar to the proofs of Claim 8.2.3 and Claim 8.2.4 respectively.

■

8.4 A New Framework for Embedded-Identity Traitor Tracing

8.4.1 Embedded-Identity Private Linear Broadcast Encryption

We introduce the notion of embedded-identity private linear broadcast encryption (EIPLBE) as a generalization of private linear broadcast encryption scheme. There are five algorithms in a EIPLBE scheme — $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec}$. The setup algorithm outputs a master secret key and a public key. The key generation algorithm is used to sample private keys for index-identity pairs (j, id). The public key encryption algorithm can be used to encrypt messages, and ciphertexts can be decrypted using any of the private keys via the decryption algorithm. In addition to these algorithms, there is also a special-encryption algorithm SplEnc. This algorithm, which uses the master secret key, can be used to encrypt messages to any index-position-value tuple (i, ℓ, b). A secret key for user (j, id) can decrypt a ciphertext for index-position-value tuple (i, ℓ, b) only if (1) $j \geq i + 1$, or (2) $(i, \ell) = (j, \perp)$ or $(i, \mathsf{id}_\ell) = (j, 1 - b)$.

Below we first provide the EIPLBE syntax, and then present the secu-

rity definitions.

Syntax A EIPLBE scheme $\mathsf{EIPLBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{SplEnc}, \mathsf{Dec})$ for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$ has the following syntax.

$\mathsf{Setup}(1^\lambda, 1^\kappa, n) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm takes as input the security parameter λ, the 'identity space' parameter κ, index space n, and outputs a master secret key msk and a public key pk.

$\mathsf{KeyGen}\,(\mathsf{msk}, \mathsf{id} \in \{0,1\}^\kappa, i \in [n]) \to \mathsf{sk}$. The key generation algorithm takes as input the master secret key, an identity $\mathsf{id} \in \{0,1\}^\kappa$ and index $i \in [n]$. It outputs a secret key sk.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The encryption algorithm takes as input a public key pk, message $m \in \mathcal{M}_\lambda$, and outputs a ciphertext ct.

$\mathsf{SplEnc}(\mathsf{key}, m, (i, \ell, b)) \to \mathsf{ct}$. The special-encryption algorithm takes as input a key key, message $m \in \mathcal{M}_\lambda$, and index-position-value tuple $(i, \ell, b) \in [n+1] \times ([\kappa] \cup \{\perp\}) \times \{0,1\}$, and outputs a ciphertext ct. (Here the scheme is said to be public key EIPLBE scheme if $\mathsf{key} = \mathsf{pk}$. Otherwise, it is said to be private key EIPLBE scheme.)

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. The decryption algorithm takes as input a secret key sk, ciphertext ct and outputs $z \in \mathcal{M}_\lambda \cup \{\perp\}$.

Correctness A EIPLBE scheme is said to be correct if there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda, \kappa, n \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, and $i \in [n+1]$, $j \in [n]$, $\mathsf{id} \in \{0,1\}^\kappa$, $\ell \in ([\kappa] \cup \{\bot\})$ and $b \in \{0,1\}$, the following holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},\mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk},\mathsf{pk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk},\mathsf{id},j) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda),$$

and if $j \geq i+1$, or $(i,\ell) = (j,\bot)$, or $(i,\mathsf{id}_\ell) = (j, 1-b)$, the following also holds

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},\mathsf{ct}) = m : \begin{array}{c}(\mathsf{msk},\mathsf{pk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{msk},\mathsf{id},j) \\ \mathsf{ct} \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i,\ell,b))\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Efficiency Let $\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}\widetilde{e}}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$ be functions. A EIPLBE scheme is said to be ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}\widetilde{e}}$, $\mathsf{T\text{-}k}$, $\mathsf{T\text{-}d}$, $\mathsf{S\text{-}c}$, $\mathsf{S\text{-}k}$)- efficient if the following efficiency requirements hold:

- The running time of $\mathsf{Setup}(1^\lambda, 1^\kappa, n)$ is at most $\mathsf{T\text{-}s}(\lambda,\kappa,n)$.
- The running time of $\mathsf{Enc}(\mathsf{pk}, m)$ is at most $\mathsf{T\text{-}e}(\lambda,\kappa,n)$.
- The running time of $\mathsf{SplEnc}(\mathsf{key}, m, (i,\ell,b))$ is at most $\mathsf{T\text{-}\widetilde{e}}(\lambda,\kappa,n)$.
- The running time of $\mathsf{KeyGen}(\mathsf{msk},\mathsf{id},i)$ is at most $\mathsf{T\text{-}k}(\lambda,\kappa,n)$.
- The running time of $\mathsf{Dec}(\mathsf{sk},\mathsf{ct})$ is at most $\mathsf{T\text{-}d}(\lambda,\kappa,n)$.
- The size of the ciphertexts is at most $\mathsf{S\text{-}c}(\lambda,\kappa,n)$.
- The size of the key is at most $\mathsf{S\text{-}k}(\lambda,\kappa,n)$.

8.4.1.1 *q*-query EIPLBE Security

Now we provide the security definitions for EIPLBE as a generalization of the PLBE q-query security Section 5.1.1. (Also, see Remark 8.1.1.)

Definition 8.4.1 (q-query Normal Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query normal hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{ct}_b) = b \;:\; \begin{array}{c}(1^\kappa, 1^n) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ m \leftarrow \mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\};\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ \mathsf{ct}_1 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (1, \perp, 0))\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $O_1(\cdot,\cdot) = \mathsf{SplEnc}(\mathsf{key}, \cdot, \cdot)$, and $O_2(\cdot,\cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot, \cdot)$ with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m, (j, \ell, \gamma))$ the index j must be equal to 1.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position j. That is, let $(j_1, \mathsf{id}_1), \ldots, (j_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$.

Definition 8.4.2 (q-query Index Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query index hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{ct}) = b \;:\; \begin{array}{c}(1^\kappa, 1^n, i) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ m \leftarrow \mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{pk});\, b \leftarrow \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i+b, \perp, 0))\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $O_1(\cdot,\cdot) = \mathsf{SplEnc}(\mathsf{key},\cdot,\cdot)$, and $O_2(\cdot,\cdot) = \mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)$ with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m,(j,\ell,\gamma))$ the index j must be equal to either i or $i+1$.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position $j \in [n]$, and no key query of the form (i,id). That is, let $(j_1,\mathsf{id}_1),\ldots,(j_k,\mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$. And, $j_a \neq i$ for any a.

Definition 8.4.3 (q-query Upper Identity Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query upper identity hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{ct}_b) = b \;:\; \begin{array}{c}(1^\kappa,1^n,i,\ell,\beta) \leftarrow \mathcal{A}(1^\lambda)\\ (\mathsf{pk},\mathsf{msk},\mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda,1^\kappa,n)\\ m \leftarrow \mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{pk})\,; b \leftarrow \{0,1\}\\ \mathsf{ct}_0 \leftarrow \mathsf{SplEnc}(\mathsf{key},m,(i+1,\bot,0))\\ \mathsf{ct}_1 \leftarrow \mathsf{SplEnc}(\mathsf{key},m,(i,\ell,\beta))\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $O_1(\cdot,\cdot) = \mathsf{SplEnc}(\mathsf{key},\cdot,\cdot)$, and $O_2(\cdot,\cdot) = \mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)$ with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m,(j,\ell,\gamma))$ the index j must be equal to either i or $i+1$.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position $j \in [n]$, and no key query of the form (i,id) such that $\mathsf{id}_\ell = 1-\beta$. That

is, let $(j_1, \mathsf{id}_1), \ldots, (j_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$. And, for every a, $(\mathsf{id}_a)_\ell \neq 1 - \beta$ or $j_a \neq i$.

Definition 8.4.4 (q-query Lower Identity Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query lower identity hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{ct}_b) = b \;:\; \begin{array}{c} (1^\kappa, 1^n, i, \ell, \beta) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ m \leftarrow \mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{pk}) \,; b \leftarrow \{0,1\} \\ \mathsf{ct}_0 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i, \perp, 0)) \\ \mathsf{ct}_1 \leftarrow \mathsf{SplEnc}(\mathsf{key}, m, (i, \ell, \beta)) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where $O_1(\cdot,\cdot) = \mathsf{SplEnc}(\mathsf{key}, \cdot, \cdot)$, and $O_2(\cdot,\cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot, \cdot)$ with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m, (j, \ell, \gamma))$ the index j must be equal to i.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position $j \in [n]$, and no key query of the form (i, id) such that $\mathsf{id}_\ell = \beta$. That is, let $(j_1, \mathsf{id}_1), \ldots, (j_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then j_a and j_b must be distinct for all $a \neq b$. And, for every a, $(\mathsf{id}_a)_\ell \neq \beta$ or $j_a \neq i$.

Definition 8.4.5 (q-query Message Hiding Security)**.** Let $q(\cdot)$ be any fixed polynomial. A EIPLBE scheme is said to satisfy q-query message hiding security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function

$\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{ct}) = b \;:\; \begin{array}{c}(1^\kappa, 1^n) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, \mathsf{msk}, \mathsf{key}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa, n) \\ (m_0, m_1) \leftarrow \mathcal{A}^{O_1(\cdot,\cdot),O_2(\cdot,\cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{SplEnc}(\mathsf{key}, m_b, (n+1, \bot, 0))\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $O_1(\cdot,\cdot) = \mathsf{SplEnc}(\mathsf{key}, \cdot, \cdot)$, and $O_2(\cdot,\cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot, \cdot)$ with the following oracle restrictions:

- SplEnc Oracle: $\mathcal{A}$ can make at most $q(\lambda)$ queries, and for each query $(m, (i, \ell, \gamma))$ the index i must be equal to $n+1$.
- KeyGen Oracle: $\mathcal{A}$ can make at most one query for each index position i. That is, let $(i_1, \mathsf{id}_1), \ldots, (i_k, \mathsf{id}_k)$ denote all the key queries made by $\mathcal{A}$, then i_a and i_b must be distinct for all $a \neq b$.

8.4.2 Building Indexed EITT from EIPLBE

In this section, we describe a simple construction for building an indexed EITT scheme from any EIPLBE scheme. The following construction and the high level proof structure is similar to the one provided in Section 5.2.

8.4.2.1 Construction

Consider an EIPLBE scheme $\mathsf{EIPLBE} = (\mathsf{EIPLBE.Setup}, \mathsf{EIPLBE.KeyGen}, \mathsf{EIPLBE.Enc}, \mathsf{EIPLBE.SplEnc}, \mathsf{EIPLBE.Dec})$ for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and identity space $\mathcal{ID} = \{\{0,1\}^\kappa\}_{\kappa \in \mathbb{N}}$. Below we provide our embedded identity TT construction with identical message and identity spaces. (Here we provide

a transformation for TT schemes with secret key tracing, but the construction can be easily extended to work in the public tracing setting if the special encryption algorithm in the underlying EIPLBE scheme is public key as well.)

$\mathsf{Setup}(1^\lambda, 1^\kappa, n) \to (\mathsf{msk}, \mathsf{pk}, \mathsf{key})$. The setup algorithm runs the EIPLBE setup as $(\mathsf{msk}, \mathsf{pk}, \mathsf{key}) \leftarrow \mathsf{EIPLBE.Setup}(1^\lambda, 1^\kappa, n)$, and outputs master secret-public-tracing key tuple $(\mathsf{msk}, \mathsf{pk}, \mathsf{key})$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, i) \to \mathsf{sk}_{i,\mathsf{id}}$. The key generation algorithm runs the EIPLBE key generation algorithm as $\mathsf{sk}_{i,\mathsf{id}} \leftarrow \mathsf{EIPLBE.KeyGen}(\mathsf{msk}, \mathsf{id}, i)$, and outputs secret key $\mathsf{sk}_{i,\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The encryption algorithm runs the EIPLBE encryption algorithm as $\mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m)$, and outputs ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to z$. The decryption algorithm runs the EIPLBE decryption algorithm as $z \leftarrow \mathsf{EIPLBE.Dec}(\mathsf{sk}, \mathsf{ct})$, and outputs z.

$\mathsf{Trace}^D(\mathsf{key}, 1^y, m_0, m_1) \to T$. Let $\epsilon = 1/y$. First, consider the Index-Trace algorithm defined in Fig. 8.8. The sub-tracing algorithm simply tests whether the decoder box uses the user key for index i where i is one of the inputs provided to Index-Trace. Now the tracing algorithm simply runs the Index-Trace algorithm for all indices $i \in [n]$, and for each index i where the Index-Trace algorithm outputs 1, the tracing algorithm adds

index i to the *index-set* of traitors T^{indx}.[1] Next, consider the $\mathsf{ID\text{-}Trace}$ algorithm defined in Fig. 8.9. The identity-tracing algorithm takes as input the index-set T^{indx} and uses the decoder box to find the identity of the particular indexed user. Next, the tracing algorithm simply runs the $\mathsf{ID\text{-}Trace}$ algorithm for all indices $i \in T^{\mathsf{indx}}$, and for each index i where the $\mathsf{ID\text{-}Trace}$ algorithm does not output $\perp$, the tracing algorithm adds the output of the $\mathsf{ID\text{-}Trace}$ algorithm to the *identity-set* of traitors T.

Concretely, the algorithm runs as follows:

- Set $T^{\mathsf{indx}} := \emptyset$. For $i = 1$ to n:
  - Compute $(b, p, q) \leftarrow \mathsf{Index\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, i)$.
  - If $b = 1$, set $T^{\mathsf{indx}} := T^{\mathsf{indx}} \cup \{(i, p, q)\}$.
- Set $T := \emptyset$. For $(i, p, q) \in T^{\mathsf{indx}}$:
  - Compute $\mathsf{id} \leftarrow \mathsf{ID\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, (i, p, q))$.
  - Set $T := T \cup \{\mathsf{id}\}$.
- Output T.

Finally, it outputs the set T as the set of traitors.

Correctness This follows directly from correctness of the underlying EIPLBE scheme.

[1] Technically, the set T^{indx} constains tuples of the form (i, p, q) where i is an index and p, q are probabilities which are the estimations of successful decryption probability at index i and $i + 1$ (respectively).

Algorithm $\mathsf{Index\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, i)$

Inputs: Key key, parameter y, messages m_0, m_1, index i
Output: $0/1$
Let $\epsilon = \lfloor 1/y \rfloor$. It sets $N = \lambda \cdot n/\epsilon$, and $\mathsf{count}_1 = \mathsf{count}_2 = 0$. For $j = 1$ to N, it computes the following:

1. It chooses $b_j \leftarrow \{0,1\}$ and computes $\mathsf{ct}_{j,1} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_{b_j}, (i, \perp, 0))$ and sends $\mathsf{ct}_{j,1}$ to D. If D outputs b_j, set $\mathsf{count}_1 = \mathsf{count}_1 + 1$, else set $\mathsf{count}_1 = \mathsf{count}_1 - 1$.
2. It chooses $c_j \leftarrow \{0,1\}$ and computes $\mathsf{ct}_{j,2} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_{c_j}, (i+1, \perp, 0))$ and sends $\mathsf{ct}_{j,2}$ to D. If D outputs c_j, set $\mathsf{count}_2 = \mathsf{count}_2 + 1$, else set $\mathsf{count}_2 = \mathsf{count}_2 - 1$.

If $\frac{\mathsf{count}_1 - \mathsf{count}_2}{N} > \frac{\epsilon}{4n}$, output $(1, \frac{\mathsf{count}_1}{N}, \frac{\mathsf{count}_2}{N})$, else output $(0, \perp, \perp)$.

Figure 8.8: $\mathsf{Index\text{-}Trace}$

Algorithm $\mathsf{ID\text{-}Trace}(\mathsf{key}, 1^y, m_0, m_1, (i, p, q))$

Inputs: Key key, parameter y, messages m_0, m_1, index i, probabilities p, q
Output: $\mathsf{id} \in \{0,1\}^\kappa$
Let $\epsilon = \lfloor 1/y \rfloor$. It sets $N = \lambda \cdot n/\epsilon$, and $\mathsf{count}_\ell = 0$ for $\ell \in [\kappa]$. For $\ell = 1$ to κ, it proceeds as follows:

1. For $j = 1$ to N, it computes the following:
   (a) It chooses $b_j \leftarrow \{0,1\}$ and computes $\mathsf{ct}_j \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_{b_j}, (i, \ell, 0))$ and sends ct_j to D. If D outputs b_j, set $\mathsf{count}_\ell = \mathsf{count}_\ell + 1$, else set $\mathsf{count}_\ell = \mathsf{count}_\ell - 1$.

Next, let id be an empty string. For $\ell = 1$ to κ, do the following:

1. If $\frac{p+q}{2} > \frac{\mathsf{count}_\ell}{N}$, set $\mathsf{id}_\ell = 0$. Else set $\mathsf{id}_\ell = 1$.

Finally, output id.

Figure 8.9: $\mathsf{Index\text{-}Trace}$

Efficiency If the scheme $\mathsf{EIPLBE} = (\mathsf{EIPLBE.Setup}, \mathsf{EIPLBE.KeyGen}, \mathsf{EIPLBE.Enc}, \mathsf{EIPLBE.SplEnc}, \mathsf{EIPLBE.Dec})$ is a EIPLBE scheme with ($\mathsf{T\text{-}s}$, $\mathsf{T\text{-}e}$, $\mathsf{T\text{-}\widetilde{e}}$, $\mathsf{T\text{-}k}$,

T-d, S-c, S-k)-efficiency, then the scheme TT = (Setup, KeyGen, Enc, Dec, Trace) is a (indexed keygen, public/private)-embedded identity tracing scheme with (T-s′, T-e′, T-k′, T-d′, T-t′, S-c′, S-k′)-efficiency, where the efficiency measures are related as follows:

- $\mathsf{T\text{-}s}'(\lambda, \kappa, n) = \mathsf{T\text{-}s}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}k}'(\lambda, \kappa, n) = \mathsf{T\text{-}k}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}e}'(\lambda, \kappa, n) = \mathsf{T\text{-}e}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}d}'(\lambda, \kappa, n) = \mathsf{T\text{-}d}(\lambda, \kappa, n)$,
- $\mathsf{T\text{-}t}'(\lambda, \kappa, n, y) = (2n + \kappa) \cdot \lambda \cdot y \cdot n$,
- $\mathsf{S\text{-}c}'(\lambda, \kappa, n) = \mathsf{S\text{-}c}(\lambda, \kappa, n)$,
- $\mathsf{S\text{-}k}'(\lambda, \kappa, n) = \mathsf{S\text{-}k}(\lambda, \kappa, n)$.

8.4.2.2 Security

In this section, we prove security of our construction. Formally, we prove the following.

Theorem 8.4.1. *If the scheme* EIPLBE = (EIPLBE.Setup, EIPLBE.KeyGen, EIPLBE.Enc, EIPLBE.SplEnc, EIPLBE.Dec) *is a 1-query secure EIPLBE scheme as per Definitions 8.4.1 to 8.4.5, then the scheme* $\mathcal{T}$ = (Setup, KeyGen, Enc, Dec, Trace) *is a secure (indexed keygen, public/private)-embedded identity tracing scheme as per Definitions 4.1.2 and 8.1.1.*

We prove the above theorem in two parts. First, we prove IND-CPA security of the above construction. Later we argue correctness of tracing to

complete the proof of security. Note that in the proof we use the fact that n is bounded by a polynomial in the security parameter. (See Remark 8.1.1.)

IND-CPA Security We would like to point out that the scheme $\mathcal{T}$ is IND-CPA secure even if the EIPLBE scheme satisfies only 0-query security. In other words, we do not need the scheme to achieve 1-query security for arguing IND-CPA security. At a high level, the proof of IND-CPA security is identical to that used for proving IND-CPA security of (standard) traitor tracing systems from (standard) private linear broadcast encryption scheme [25]. Below we provide a high level sketch.

Lemma 8.4.1. *If the scheme* EIPLBE *is a 0-query secure EIPLBE scheme as per Definitions 8.4.1, 8.4.2 and 8.4.5, then the scheme* $\mathcal{T}$ *is an IND-CPA secure (indexed keygen, public/private)-embedded identity tracing scheme as per Definition 8.1.1.*

Proof. We will construct a sequence of $2n + 4$ hybrid experiments to prove IND-CPA security. The first experiment, that is Hybrid H_0, is exactly the IND-CPA game.

Hybrid H_0 : In this experiment, the challenger sends public key pk, receives m_0, m_1 from $\mathcal{A}$ and sends $\mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m_0)$ to $\mathcal{A}$.

Hybrid $H_{i,b}$ (**for** $i \in [n+1], b \in \{0,1\}$) : This experiment is identical to the IND-CPA experiment, except that the adversary, after sending challenge

messages m_0, m_1, receives $\mathsf{ct} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_b, (i, \perp, 0))$.

Hybrid H_1 : In this experiment, the challenger sends public key pk, receives m_0, m_1 from $\mathcal{A}$ and sends $\mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m_1)$ to $\mathcal{A}$.

For any PPT adversary $\mathcal{A}$, let $p_{\mathcal{A},x}(\cdot)$ be a function of λ that denotes the probability of $\mathcal{A}$ outputting 0 in Hybrid H_x. Note that $p_{\mathcal{A},0} - p_{\mathcal{A},1}$ is the advantage of $\mathcal{A}$ in the IND-CPA security game.

Claim 8.4.1. *If the scheme* EIPLBE *is a 0-query normal hiding secure EIPLBE scheme as per Definition 8.4.1, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $b \in \{0,1\}$, $|p_{\mathcal{A},b} - p_{\mathcal{A},1,b}| \leq negl(\lambda)$.*

This follows from 0-query normal hiding security of EIPLBE.

Claim 8.4.2. *If the scheme* EIPLBE *is a 0-query index hiding secure EIPLBE scheme as per Definition 8.4.2, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $(i, b) \in [n] \times \{0,1\}$, $|p_{\mathcal{A},i,b} - p_{\mathcal{A},i+1,b}| \leq negl(\lambda)$.*

This follows from 0-query index hiding security of EIPLBE.

Claim 8.4.3. *If the scheme* EIPLBE *is a 0-query message hiding secure EIPLBE scheme as per Definition 8.4.5, then for any PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|p_{\mathcal{A},n+1,0} - p_{\mathcal{A},n+1,1}| \leq negl(\lambda)$.*

This follows from 0-query message hiding security of EIPLBE.

Combining the above claims, we get proof of Lemma 8.4.1.

■

Correctness of Tracing Next, we show that the false trace probability is bounded by a negligible function, and the correct trace probability is close to the probability of $\mathcal{A}$ outputting an ϵ-successful decoding box for some non-negligible ϵ.

First, we introduce some notations. Fix some public-master secret key pair $(\mathsf{pk}, \mathsf{msk})$. Given any pirate decoder box D and messages m_0, m_1, for any $i \in [n+1]$, $\ell \in [\kappa]$, let

$$p^D_{i,\perp} = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_b, (i, \perp, 0))]$$

$$p^D_{i,\ell} = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{EIPLBE.SplEnc}(\mathsf{key}, m_b, (i, \ell, 0))]$$

$$p^D_{\mathsf{nrml}} = \Pr[D(\mathsf{ct}) = b \ : \ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{EIPLBE.Enc}(\mathsf{pk}, m_b)]$$

where the probability is taken over random coins of decoder D as well as the randomness used during encryption.

False Trace Probability First, we show that the tracing algorithm never falsely accuses any user with non-negligible probability. Formally, we prove the following.

Theorem 8.4.2. *If the scheme* EIPLBE *is a 1-query secure EIPLBE scheme as per Definitions 8.4.1 to 8.4.5, then for every PPT adversary* $\mathcal{A}$*, polynomial*

$q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$,

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq negl(\lambda),$$

where $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ is as defined in Definition 4.1.2.

Proof. Let $S \subseteq [n] \times \{0,1\}^\kappa$ be the set of index-identity pairs queried by the adversary $\mathcal{A}$ for secret keys, $S_{\mathsf{indx}} \subseteq [n]$ be the set of indices queried by the adversary $\mathcal{A}$ for secret keys, and let D be the decoder box output by $\mathcal{A}$.

In the sequel we skip the dependence of $\epsilon(\cdot)$ on λ for simplicity of notation. For $i \in [n], \ell \in [\kappa]$, we define events

$$\begin{aligned}
\mathsf{Diff\text{-}Adv}_i^D &: p_{i,\perp}^D - p_{i+1,\perp}^D > \epsilon/8n \\
\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}}^D &: p_{i,\perp}^D - p_{i,\ell}^D > \epsilon/16n \\
\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{upr}}^D &: p_{i,\ell}^D - p_{i+1,\perp}^D > \epsilon/16n \\
\mathsf{Diff\text{-}Adv}^D &: \bigvee_{i \in [n] \setminus S_{\mathsf{indx}}} \mathsf{Diff\text{-}Adv}_i^D \bigvee_{\substack{(i,\mathsf{id}) \in S,\ \ell \in [\kappa] \\ \text{s.t. } \mathsf{id}_\ell = 1}} \mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}}^D \bigvee_{\substack{(i,\mathsf{id}) \in S,\ \ell \in [\kappa] \\ \text{s.t. } \mathsf{id}_\ell = 0}} \mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{upr}}^D
\end{aligned}$$

For simplicity of notation, we will drop dependence on decoder D whenever clear from context. Next, note that the probability of the event *false trace* can be rewritten (using union bound) as follows by conditioning on the events defined above

$$\Pr[\mathsf{Fal\text{-}Tr}] \leq \Pr\left[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}\right] + \sum_{i \in [n]} \Pr\left[i \notin S_{\mathsf{indx}} \wedge \mathsf{Diff\text{-}Adv}_i\right]$$

$$+ \sum_{(i,\ell)\in[n]\times[\kappa]} \Pr\left[\begin{array}{c} \exists \mathsf{id} \in \{0,1\}^{\kappa} \text{ such that} \\ (i,\mathsf{id}) \in S \wedge \left(\begin{array}{c} (\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}} \wedge \mathsf{id}_\ell = 1) \vee \\ (\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{upr}} \wedge \mathsf{id}_\ell = 0) \end{array}\right) \end{array}\right].$$

We will show that each of these terms is bounded by a negligible function. We start by bounding the first term.

Lemma 8.4.2. *For every PPT adversary $\mathcal{A}$, there exists a negligible function $negl_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}] \leq negl_1(\lambda).$$

Proof. The proof of this lemma follows from Chernoff bounds, and is similar to those provided in [82, Lemma 4.4] and [79, Lemma 5.3]. Here we sketch the high level idea.

Note that event $\mathsf{Fal\text{-}Tr}$ occurs iff the tracing algorithm outputs a user identity which was not key queried by the adversary. Recall that the tracing algorithm takes a two-step approach. It proceeds by first tracing the key indices of the corrupted keys, and then it traces the corresponding identity. Now there are two sources of error in incorrect tracing. First, during step one of tracing the algorithm might incorrectly include some index $i \notin S_{\mathsf{indx}}$ in the index-set of traitors T^{indx}. Second, during step two it may output a non-corrupt identity id for some index $i \in S_{\mathsf{indx}}$, that is for some $i \in S_{\mathsf{indx}}$ the $\mathsf{ID\text{-}Trace}$ algorithm traces the id incorrectly at at least one bit position. Thus, we could write the following (using union bound), where sets T and T^{indx} are

as defined in the description of the tracing algorithm,

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}] \leq \sum_{i\in[n]} \Pr\left[\begin{array}{c}\mathsf{Fal\text{-}Tr} \wedge i \notin S_{\mathsf{indx}} \\ \wedge(\exists\ p,q : (i,p,q) \in T^{\mathsf{indx}})\end{array} \mid \overline{\mathsf{Diff\text{-}Adv}}\right]$$
$$+ \sum_{(i,\ell)\in[n]\times[\kappa]} \Pr\left[\begin{array}{c}\mathsf{Fal\text{-}Tr} \wedge \exists \mathsf{id}, \widetilde{\mathsf{id}} : (i,\mathsf{id}) \in S \\ \wedge \widetilde{\mathsf{id}} \in T \wedge \mathsf{id}_\ell \neq \widetilde{\mathsf{id}}_\ell\end{array} \mid \overline{\mathsf{Diff\text{-}Adv}}\right].$$

In the above expression, the first term on the right side bounds the type 1 error (i.e., faulty step one tracing) and the second term bounds the type 2 error (i.e., faulty step two tracing).

Now let us analyze the first term. Note that if event $\overline{\mathsf{Diff\text{-}Adv}}$ occurs then it implies that for every $i \notin S_{\mathsf{indx}}$ event $\overline{\mathsf{Diff\text{-}Adv}_i}$ occurred. Thus, it must hold that for every $i \in [n]$

$$\Pr\left[i \notin S_{\mathsf{indx}} \wedge (\exists\ p,q : (i,p,q) \in T^{\mathsf{indx}}) \mid \overline{\mathsf{Diff\text{-}Adv}}\right] \leq 2^{-O(\lambda)}.$$

This follows from a Chernoff bound since $\overline{\mathsf{Diff\text{-}Adv}_i}$ states that $p_{i,\perp} - p_{i+1,\perp} \leq \epsilon/8n$ and event $(\exists\, p,q : (i,p,q) \in T^{\mathsf{indx}})$ suggests that $\hat{p}_{i,\perp} - \hat{p}_{i+1,\perp} > \epsilon/4n$ where $\hat{p}$ denotes the corresponding estimate computed by the tracing algorithm.

Next, let us analyze the second term. For a fixed (i,ℓ), the probability term corresponds to the event that the $\mathsf{ID\text{-}Trace}$ algorithm outputs the traitor identity $\widetilde{\mathsf{id}}$ such that $\mathsf{id}_\ell \neq \widetilde{\mathsf{id}}_\ell$ where the adversary makes a key query for index-identity pair (i,id). (Recall that the adversary is allowed to receive at most one key per index. See Definition 4.1.2.) Concretely, by conditioning on the event $\overline{\mathsf{Diff\text{-}Adv}}$ we get that for every $(i,\mathsf{id}) \in S, \ell \in [\kappa]$, event $\overline{\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{X}}}$ always occurs where $\mathrm{X} = \mathrm{lwr}$ if $\mathsf{id}_\ell = 1$ else $\mathrm{X} = \mathrm{upr}$. Thus, it must hold that

for every $(i, \mathsf{id}) \in S, \ell \in [\kappa]$

$$\Pr\left[\exists \mathsf{id}, \widetilde{\mathsf{id}} : (i, \mathsf{id}) \in S \wedge \widetilde{\mathsf{id}} \in T \wedge \mathsf{id}_\ell \neq \widetilde{\mathsf{id}}_\ell \mid \overline{\mathsf{Diff\text{-}Adv}}\right] \leq 2^{-O(\lambda)}.$$

For simplicity, fix (i, id, ℓ) and let $\mathsf{id}_\ell = 1$. Then the above statement follows from a Chernoff bound since we know that event $\overline{\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}}}$ occurs, thus we have that $p_{i,\perp} - p_{i,\ell} \leq \epsilon/16n$ and event $\widetilde{\mathsf{id}} \in T \wedge \widetilde{\mathsf{id}}_\ell = 0$ suggests that $\hat{p}_{i,\perp} - \hat{p}_{i,\ell} > \epsilon/8n$ where $\hat{p}$ denotes the corresponding estimate computed by the tracing algorithm.

Therefore, combining all the above claims we get that

$$\Pr[\mathsf{Fal\text{-}Tr} \mid \overline{\mathsf{Diff\text{-}Adv}}] \leq n \cdot 2^{-O(\lambda)} + n \cdot \kappa \cdot 2^{-O(\lambda)} = \mathrm{negl}_1(\lambda).$$

■

Lemma 8.4.3. *If the scheme* EIPLBE *is a 1-query index hiding secure EIPLBE scheme as per Definition 8.4.2, then for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $negl_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$ and $i \in [n]$,*

$$\Pr[i \notin S_{\mathsf{indx}} \wedge \mathsf{Diff\text{-}Adv}_i] \leq negl_2(\lambda),$$

where n is the index bound chosen, and S_{indx} is the set of indices queried by $\mathcal{A}$.

Proof. The proof of this lemma is similar to those provided in [82, Lemma 4.5] and [79, Lemma 5.4]. ■

Lemma 8.4.4. *If the scheme* EIPLBE *is a 1-query lower and upper identity hiding secure EIPLBE scheme as per Definitions 8.4.3 and 8.4.4, then for every PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible function $\epsilon(\cdot)$, there exists a negligible function $negl_3(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$ and $i \in [n], \ell \in [\kappa]$,*

$$\Pr\left[\exists \mathsf{id} \in \{0,1\}^{\kappa} \text{ s.t. } (i,\mathsf{id}) \in S \wedge \begin{pmatrix} (\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{lwr}} \wedge \mathsf{id}_\ell = 1) \vee \\ (\mathsf{Diff\text{-}Adv}_{i,\ell,\mathrm{upr}} \wedge \mathsf{id}_\ell = 0) \end{pmatrix}\right] \leq negl_3(\lambda),$$

where n is the index bound chosen, and S is the set of index-identity pairs queried by $\mathcal{A}$.

Proof. Suppose, on the contrary, there exists a PPT adversary $\mathcal{A}$, polynomial $q(\cdot)$ and non-negligible functions $\epsilon(\cdot), \delta(\cdot)$ such that for all $\lambda \in \mathbb{N}$ satisfying $\epsilon(\lambda) > 1/q(\lambda)$, there exists an $i^* \in [n], \ell^* \in [\kappa]$ s.t.

$$\Pr\left[\exists \mathsf{id} \in \{0,1\}^{\kappa} \text{ s.t. } (i^*,\mathsf{id}) \in S \wedge \begin{pmatrix} (\mathsf{Diff\text{-}Adv}_{i^*,\ell^*,\mathrm{lwr}} \wedge \mathsf{id}_{\ell^*} = 1) \vee \\ (\mathsf{Diff\text{-}Adv}_{i^*,\ell^*,\mathrm{upr}} \wedge \mathsf{id}_{\ell^*} = 0) \end{pmatrix}\right] \geq \delta(\lambda).$$

Then we can use $\mathcal{A}$ to build a PPT reduction algorithm $\mathcal{B}$ that breaks the upper/lower identity hiding security property of EIPLBE. The reduction algorithm $\mathcal{B}$ first receives $1^n, 1^\kappa$ from the adversary. It chooses a random index $i \leftarrow [n]$, position $\ell \in [\kappa]$, and value $b \in \{0,1\}$, and sends the challenge index-position-value tuple $(i, \ell, 0)$ and $(1^n, 1^\kappa)$ to the EIPLBE challenger.[2] (In other words, the reduction algorithm randomly guesses the index-position pair

[2]Note that both n and κ are outputted in unary, thus they are some polynomials in the security parameter.

(i^*, ℓ^*) as well as if $b = 0$ it interacts with EIPLBE lower identity hiding challenger, otherwise if $b = 1$ it interacts with EIPLBE upper identity hiding challenger.) It then receives the EIPLBE public key pk from the challenger, which it sends to $\mathcal{A}$. The adversary $\mathcal{A}$ then queries for secret keys. If $\mathcal{A}$ key queries for index-identity pair (j, id) where $j = i$ and $\mathsf{id}_\ell = b$, then $\mathcal{B}$ aborts and sends a random guess to the EIPLBE challenger. Else, on key query for (j, id) from $\mathcal{A}$, reduction algorithm $\mathcal{B}$ forwards (j, id) to the EIPLBE challenger and forwards the challenger's response to the adversary. After all key queries, the adversary outputs a decoding box D and messages m_0, m_1 to $\mathcal{B}$. $\mathcal{B}$ then chooses two bits α, β uniformly at random, i.e. $\alpha, \beta \leftarrow \{0, 1\}$. Next, $\mathcal{B}$ sends message m_α as its challenge message, and receives challenge ciphertext ct^* from EIPLBE challenger. It also queries the EIPLBE challenger for a special-encryption of m_α for index-position-value tuple $(i, \ell, 0)$ if $\beta = 0$, else for $(i + b, \perp, 0)$. Let ct be the challenger's response. Finally, $\mathcal{B}$ runs decoder box D on ct and ct^* independently, and if $D(\mathsf{ct}) = D(\mathsf{ct}^*)$, it outputs $b' = \beta$, else it outputs $b' = 1 - \beta$ as its guess.

First, note that $\mathcal{B}$ is an admissible adversary in the upper/lower identity hiding security game (if $b = 0$ then 'lower', else 'upper' respectively). This is because $\mathcal{B}$ does not query the challenger for secret key on index-identity pair (j, id) such that $j = i$ and $\mathsf{id}_\ell = b$. Additionally, it only makes a single special-encryption query on index-position-value tuple $(i, \ell, 0)$ or $(i + b, \perp, 0)$. Finally, by an analysis similar to that in [82, Lemma 4.1, 4.5] and [79, Lemma 5.4], it follows that the advantage of the reduction algorithm is at least $\frac{\delta}{2\kappa n} \left(\frac{\epsilon}{16n}\right)^2$.

Thus, the lemma follows. ■

From the above lemmas, it follows that the probability of false trace is at most $\mathrm{negl}_1(\lambda) + n \cdot \mathrm{negl}_2(\lambda) + n \cdot \kappa \cdot \mathrm{negl}_3(\lambda)$, thus theorem follows.

■

Correct Trace Probability Now we show that whenever the adversary outputs a good decoder, then with all but negligible probability the tracing algorithm outputs a non-empty set T. Combining this with Theorem 8.4.2, we get that the tracing algorithm correctly traces. Formally, we show the following.

Theorem 8.4.3. *If the scheme* EIPLBE *is a 1-query secure EIPLBE scheme as per Definitions 8.4.1 to 8.4.5, then for every PPT adversary* $\mathcal{A}$*, polynomial* $q(\cdot)$ *and non-negligible function* $\epsilon(\cdot)$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *satisfying* $\epsilon(\lambda) > 1/q(\lambda)$*,*

$$\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\lambda) - negl(\lambda)$$

where $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\epsilon}(\cdot)$ *and* $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\epsilon}(\cdot)$ *are as defined in Definition 4.1.2.*

Proof. Let us start by analyzing the probability that tracing algorithm outputs a non-empty set T. First, we know that if event $\mathsf{Good\text{-}Decoder}$ occurs, then $p^D_{\mathsf{nrml}} \geq 1/2 + \epsilon$ for some non-negligible ϵ. Next, let $S_{\mathsf{indx}} \subseteq [n]$ be the set

of indices $i \in [n]$ such that $p^D_{i,\perp} - p^D_{i+1,\perp} > \epsilon/2n$. By using Chernoff bounds similar to that in Lemma 8.4.2, we get that

$$\forall\ i \in S_{\mathsf{indx}}, \quad \Pr\left[\hat{p}^D_{i,\perp} - \hat{p}^D_{i+1,\perp} < \epsilon/4n\right] \leq 2^{-O(\lambda)} = \mathrm{negl}_1(\lambda), \tag{8.18}$$

where $\hat{p}$ denotes the corresponding estimate computed by the tracing algorithm.

Note that by 1-query message hiding security of the underlying EIPLBE scheme, we have that $p^D_{n+1,\perp} \leq 1/2 + \mathrm{negl}_2(\lambda)$ for some negligible function $\mathrm{negl}_2(\cdot)$. Also, by 1-query normal hiding security, we have that $p^D_{\mathsf{nrml}} - p^D_{1,\perp} \leq \mathrm{negl}_3(\lambda)$ for some negligible function $\mathrm{negl}_3(\cdot)$. Thus, we can write that

$$p^D_{1,\perp} - p^D_{n+1,\perp} \geq \epsilon - \mathrm{negl}_2(\lambda) - \mathrm{negl}_3(\lambda) > \epsilon/2.$$

Given this we can conclude that the set S_{indx} (as defined above) must be non-empty whenever event Good-Decoder occurs. Combining this with Eq. (8.18), we get that if event Good-Decoder occurs then with all-but-negligible probability

$$T^{\mathsf{indx}} \neq \emptyset, \quad \text{and} \quad \forall\ (i,p,q) \in T^{\mathsf{indx}} :\ p - q > \frac{\epsilon}{4n}$$

where T^{indx} is as defined in the tracing algorithm.

Looking back at Fig. 8.9, we observe that for every tuple (i,p,q) the ID-Trace algorithm always outputs some identity id. This is because the algorithm simply checks for every $\ell \in [\kappa]$, either $p^D_{i,\ell} > (p+q)/2$ and the algorithm sets $\mathsf{id}_\ell = 1$, otherwise it sets $\mathsf{id}_\ell = 0$. Thus, this implies the following:

$$T^{\mathsf{indx}} \neq \emptyset \implies T \neq \emptyset.$$

Therefore, we can write the following

$$\Pr[T \neq \emptyset] \geq (1 - n \cdot \mathrm{negl}_1(\lambda)) \cdot \Pr\text{-}\mathsf{G}\text{-}\mathsf{D}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G}\text{-}\mathsf{D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda).$$

Finally, combining with Theorem 8.4.2, we get that

$$\Pr\text{-}\mathsf{Cor}\text{-}\mathsf{Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \Pr\text{-}\mathsf{G}\text{-}\mathsf{D}_{\mathcal{A},\epsilon}(\lambda) - \mathrm{negl}(\lambda).$$

This concludes the proof. ■

8.5 Building EIPLBE from ABE and Mixed FE

We now briefly discuss how to build EIPLBE with ciphertext size poly-logarithmic in the index bound n. For this construction, we rely on the notion of mixed FE as described in Section 4.2.

In Section 5.4, we described a PLBE construction from a KP-ABE scheme and a mixed FE scheme for comparisons. It turns out that the same transformation can also be used to build as EIPLBE scheme if we use the following (more expressive) function family:

$$\{f_{y^*,\ell,b}(y, \mathsf{id}) = 1 \text{ iff } y > y^* \text{ or } (y, \mathsf{id}_\ell) = (y^*, 1-b)\}_{y^*,\ell,b}\,.$$

Since the functions in the above function family can be represented by a polynomial sized branching program (, or a log-depth circuit), thus we could instantiate the above construction using our mixed FE construction from Chapter 7. Hence, combining this with existing LWE-based KP-ABE schemes [76, 22], we obtain an LWE-based EIPLBE scheme with succinct ciphertexts, thereby

building a collusion-resistant compact EITT scheme via the transformations described in previous sections.

Appendices

Appendix A

ABE and Mixed FE to PLBE: Preserving perfect correctness

In this chapter, we give an alternate construction for constructing PLBE such that if the underlying ABE scheme achieves *perfect* correctness, then so does the PLBE scheme, even if the mixed FE scheme is *not* perfectly correct. Since existing ABE schemes [76, 22] can be made perfectly correct by appropriately truncating noise distributions used, this gives a pathway to get perfect correctness under LWE. Note that the construction described in Section 5.4.1 only achieves perfect correctness when both underlying ABE and mixed FE schemes are perfectly correct. This is because the policy circuit is the mixed FE decryption circuit, and thus in order to guarantee perfect correctness, we need the minimum requirement that the mixed FE normal ciphertexts always decrypt to 1. Below we give the main idea to obtain perfect correctness.

Outline At a very high level, the idea is to encrypt the message m under two independent ABE systems such that at least one of the ciphertext components can always be decrypted to obtain the underlying message. To this end, during setup we sample two ABE key pairs $(\mathsf{abe.pp}_b, \mathsf{abe.msk}_b)$ (for

$b \in \{0,1\}$) and a mixed FE key pair $(\mathsf{mixed.pp}, \mathsf{mixed.msk})$. To generate the secret key for the ith user, we generate a mixed FE secret key $\mathsf{mixed.sk}_i$ for message i, and later compute two ABE keys $\mathsf{abe.sk}_{i,0}, \mathsf{abe.sk}_{i,1}$ for predicates $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$ and $\overline{\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)}$ using $\mathsf{abe.msk}_0, \mathsf{abe.msk}_1$, respectively. Here $\overline{\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)}$ denotes the circuit that first decrypts the input using key $\mathsf{mixed.sk}_i$ and later applies a "not" gate (i.e., outputs the complement). Now the PLBE ciphertexts will consist of two parts, one for each ABE subsystem. For PLBE normal encryption, one computes two ciphertexts ct_b (for $b \in \{0,1\}$) as encryptions of message m under attributes $\mathsf{ct}_{\mathsf{attr}}$ using parameters $\mathsf{abe.pp}_b$, where $\mathsf{ct}_{\mathsf{attr}}$ is computed as before. Now for encrypting a message to index i, the encryption algorithm behaves differently in that it computes ct_0 as before, but ct_1 will now be an encryption of message 0 under the same attributes. The reason for not encrypting the message m in the second component of the index ciphertext becomes clear while proving security.

Now for arguing perfect correctness, we observe that it should be the case that $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}})$ equals either 0 or 1. Thus, at least one of the PLBE normal ciphertext components could be correctly decrypted. Note that for such an argument we only require that ABE be perfectly correct. Next, the security proof is similar to that in Section 5.4.3, except that when arguing normal hiding security of our construction we need to rely on both the ABE security as well as the weak accept indistinguishability property of the mixed FE scheme. The main idea is that by correctness of the functional encryption

scheme, we can say that with all but negligible probability the attribute used in the second component of the challenge ciphertext is not satisfied by any of the ABE keys queried. Thus, as our first hybrid argument, we could use ABE security to switch the second challenge ciphertext component to an encryption of 0 instead of message m. The remaining proof is identical to as before, with the only modification being that the reduction algorithm needs to generate the ABE keys for the second component on its own during the entire reduction. Below we describe our construction $\mathsf{PLBE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Enc\text{-}index}, \mathsf{Dec})$ for messages spaces $\{\mathcal{M}_\kappa\}_\kappa$ in detail.

A.1 Construction

Let $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Dec})$ be a KP-ABE scheme for a set of attribute spaces $\{\mathcal{X}_\kappa\}_\kappa$, predicate classes $\{\mathcal{C}_\kappa\}_\kappa$, and message spaces $\{\mathcal{M}_\kappa\}_\kappa$, and let $\mathsf{Mixed\text{-}FE} = (\mathsf{Mixed.Setup}, \mathsf{Mixed.Enc}, \mathsf{Mixed.SK\text{-}Enc}, \mathsf{Mixed.KeyGen}, \mathsf{Mixed.Dec})$ be a mixed FE scheme, for function classes $\{\mathcal{F}_\kappa\}_\kappa$ and message space $\{\mathcal{I}_\kappa\}_\kappa$, with ciphertexts of length $\ell(\lambda, \kappa)$. For every n, let $\kappa = \kappa(n)$ be the lexicographically smallest functionality index such that every string of length $\log(n)$ can be uniquely represented in message space $\mathcal{I}_\kappa$ (i.e., $\{0,1\}^{\log(n)} \subseteq \mathcal{I}_\kappa$), and function class $\mathcal{F}_\kappa$ contains the "comparison" ($>$) operator. Also, let $\widetilde{\kappa} = \widetilde{\kappa}(\lambda, \kappa)$ be the lexicographically smallest functionality index such that every string of length $\ell(\lambda, \kappa)$ can be uniquely represented in attribute class $\mathcal{X}_{\widetilde{\kappa}}$ (i.e., $\{0,1\}^{\ell(\lambda,\kappa)} \subseteq \mathcal{X}_{\widetilde{\kappa}}$), and $\mathcal{C}_{\widetilde{\kappa}}$ contains a mixed FE decryption circuit (as well its complement circuit) corresponding to functionality index κ. Below

we describe our construction.

- $\mathsf{Setup}(1^\lambda, 1^n) \to \big(\mathsf{pp}, \mathsf{msk}, \{\mathsf{sk}_i\}_{i\le n}\big)$. The setup algorithm runs $\mathsf{ABE.Setup}$ and $\mathsf{Mixed.Setup}$ to generate ABE and mixed FE public parameters and master secret key as $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$ and $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$. Next, it runs $\mathsf{Mixed.KeyGen}$ to generate n mixed FE secret keys $\mathsf{mixed.sk}_i$ as

$$\forall\ i \le n, \quad \mathsf{mixed.sk}_i \leftarrow \mathsf{Mixed.KeyGen}(\mathsf{mixed.msk}, i).$$

  Let $C^0_{\mathsf{mixed.sk}_i}$ denote the circuit $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)$, and let $C^1_{\mathsf{mixed.sk}_i}$ denote the circuit $\overline{\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \cdot)}$; i.e., $C^1_{\mathsf{mixed.sk}_i}$ is the $\mathsf{Mixed\text{-}FE}$ decryption circuit with key $\mathsf{mixed.sk}_i$ hardwired and a "not" gate applied on the output of decryption. Next, it computes $2n$ ABE secret keys $\mathsf{abe.sk}_{i,b}$ as

$$\forall\ i \le n, b \in \{0,1\}, \quad \mathsf{abe.sk}_{i,b} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}_b, C^b_{\mathsf{mixed.sk}_i}).$$

  Finally, it sets $\mathsf{pp} = (\mathsf{abe.pp}_0, \mathsf{abe.pp}_1, \mathsf{mixed.pp})$, $\mathsf{msk} = (\mathsf{abe.msk}_0, \mathsf{abe.msk}_1, \mathsf{mixed.msk})$, and $\mathsf{sk}_i = (\mathsf{abe.sk}_{i,0}, \mathsf{abe.sk}_{i,1})$ for $i \le n$.

- $\mathsf{Enc}(\mathsf{pp}, m) \to \mathsf{ct}$. Let $\mathsf{pp} = (\mathsf{abe.pp}_0, \mathsf{abe.pp}_1, \mathsf{mixed.pp})$. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$. Next, it encrypts message m as $\mathsf{ct}_b \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_b, \mathsf{ct}_{\mathsf{attr}}, m)$ for $b \in \{0,1\}$ and outputs ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.

- $\mathsf{Enc\text{-}index}(\mathsf{msk}, m, i) \to \mathsf{ct}$. Let $\mathsf{msk} = (\mathsf{abe.msk}_0, \mathsf{abe.msk}_1, \mathsf{mixed.msk})$ and comp_i denote the comparison function $\overset{?}{>} i$, i.e., $\mathsf{comp}_i(x) = 1$ iff $x >$

i. The encryption algorithm first computes $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.SK\text{-}Enc}(\mathsf{mixed.msk}, \mathsf{comp}_i)$. Next, it encrypts message m as $\mathsf{ct}_0 \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_0, \mathsf{ct}_{\mathsf{attr}}, m)$ and $\mathsf{ct}_1 \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_1, \mathsf{ct}_{\mathsf{attr}}, 0)$ and outputs ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow m$ or $\perp$. Let $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$ and $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$. The decryption algorithm runs $\mathsf{ABE.Dec}$ on ciphertexts ct_b using key sk_b as $y_b = \mathsf{ABE.Dec}(\mathsf{sk}_b, \mathsf{ct}_b)$ for $b \in \{0, 1\}$. If $y_0 \neq \perp$, it outputs y_0. Otherwise, it sets y_1 as the output of decryption.

A.2 Correctness

For all $\lambda, n \in \mathbb{N}$, message $m \in \mathcal{M}_\lambda$, public parameters and master secret keys $(\mathsf{abe.pp}_b, \mathsf{abe.msk}_b) \leftarrow \mathsf{ABE.Setup}(1^\lambda, 1^{\widetilde{\kappa}})$ (for $b \in \{0, 1\}$), $(\mathsf{mixed.pp}, \mathsf{mixed.msk}) \leftarrow \mathsf{Mixed.Setup}(1^\lambda, 1^\kappa)$, the secret keys $\mathsf{sk}_{i,b}$ for $i \leq n, b \in \{0, 1\}$ are simply the ABE keys $\mathsf{abe.sk}_{i,b} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{abe.msk}_b, C^b_{\mathsf{mixed.sk}_i})$. For any index $i \leq n$, consider the following two cases:

1. **Normal encryption.** For any ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$ computed as $\mathsf{ct}_b \leftarrow \mathsf{ABE.Enc}(\mathsf{abe.pp}_b, \mathsf{ct}_{\mathsf{attr}}, m)$ for $b \in \{0, 1\}$, where $\mathsf{ct}_{\mathsf{attr}} \leftarrow \mathsf{Mixed.Enc}(\mathsf{mixed.pp})$, we know that either $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = 1$ or $\mathsf{Mixed.Dec}(\mathsf{mixed.sk}_i, \mathsf{ct}_{\mathsf{attr}}) = 0$. In other words, $C^b_{\mathsf{mixed.sk}_i}(\mathsf{ct}_{\mathsf{attr}}) = 1$ for some bit $b \in \{0, 1\}$. Therefore, by perfect correctness of ABE scheme, we have that for some bit $b \in \{0, 1\}$, $\mathsf{ABE.Dec}(\mathsf{abe.sk}_{i,b}, \mathsf{ct}_b) = m$. Therefore, the PLBE decryption algorithm always decrypts the normal ciphertexts correctly.

2. **Index encryption.** This is identical to the argument provided in Section 5.4.2. Note that perfect correctness for PLBE only requires perfect decryption in the case of normal encryption. Thus, it is sufficient to prove statistical correctness in the case of index encryption.

Therefore, the PLBE scheme is perfectly correct.

A.3 Security

The proof of security is almost identical to that provided in Section 5.4.3, except that to argue normal hiding security of our construction, we first need to use ABE security of the auxiliary subsystem (for $b = 1$) and the statistical correctness property of the underlying Mixed FE scheme simultaneously to switch the encryption of challenge message m^* to 0. The rest of the proof is identical.

Index

Bibliography

[1] Michel Abdalla, Alexander W. Dent, John Malone-Lee, Gregory Neven, Duong Hieu Phan, and Nigel P. Smart. Identity-based traitor tracing. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, pages 361–376, 2007.

[2] Shweta Agrawal, Sanjay Bhattacherjee, Duong Hieu Phan, Damien Stehlé, and Shota Yamada. Efficient public trace and revoke from standard assumptions: Extended abstract. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2277–2293, 2017.

[3] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical ibe. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 98–115, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Con-*

ference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II, pages 500–518, 2013.

[5] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *Annual Cryptology Conference*, 2016.

[6] Miklós Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, pages 1–9, 1999.

[7] Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee. Cryptanalysis of indistinguishability obfuscations of circuits over ggh13. Cryptology ePrint Archive, Report 2016/1003, 2016.

[8] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.

[9] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. Cryptology ePrint Archive, Report 2011/613, 2011. http://eprint.iacr.org/2011/613.

[10] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, Francois-Xavier Standaert, and Yu Yu. Leftover hash lemma,

revisited. In *Annual Cryptology Conference*, pages 1–20. Springer, 2011.

[11] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[13] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, 1986.

[14] Olivier Billet and Duong Hieu Phan. Efficient traitor tracing from collusion secure codes. In *Information Theoretic Security, Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings*, pages 171–182, 2008.

[15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190, 2015.

[16] Carlo Blundo and Antonella Cresti. Space requirements for broadcast

encryption. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 287–298. Springer, 1994.

[17] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. Practical signatures from standard assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 461–485. Springer, 2013.

[18] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *International conference on the theory and applications of cryptographic techniques*, pages 506–522. Springer, 2004.

[19] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 338–353, 1999.

[20] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil Pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[21] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *International Workshop on Public Key Cryptography*, 2011.

[22] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 533–556, 2014.

[23] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 410–428, 2013.

[24] Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 501–510, 2008.

[25] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[26] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *Proceedings of the 8th conference on The-*

ory of cryptography, TCC'11, pages 253–273, Berlin, Heidelberg, 2011. Springer-Verlag.

[27] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 211–220, 2006.

[28] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014.

[29] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 480–499, 2014.

[30] Allan Borodin, Danny Dolev, Faith E. Fich, and Wolfgang J. Paul. Bounds for width two branching programs. *SIAM J. Comput.*, 15(2):549–560, 1986.

[31] Zvika Brakerski, Craig Gentry, Shai Halevi, Tancrède Lepoint, Amit Sahai, and Mehdi Tibouchi. Cryptanalysis of the quadratic zero-testing of GGH. *IACR Cryptology ePrint Archive*, 2015.

[32] Zvika Brakerski and Oded Goldreich. From absolute distinguishability to positive distinguishability. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 141–155. Springer, 2011.

[33] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584, 2013.

[34] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 535–564. Springer, 2018.

[35] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 306–324, 2015.

[36] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained prfs (and more) from LWE. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, pages 264–302, 2017.

[37] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Obfuscating conjunctions under entropic ring lwe. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 2016.

[38] Caesar cipher. Caesar cipher — Wikipedia, the free encyclopedia. [Online; accessed 1-October-2019].

[39] Ran Canetti and Yilei Chen. Constraint-hiding constrained prfs for nc1 from lwe. In *EUROCRYPT*, 2017.

[40] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 523–552, 2010.

[41] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 542–558, 2005.

[42] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from lwe made simple and attribute-based. In *Theory of Cryptography Conference*, pages 341–369. Springer, 2018.

[43] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. Ggh15 beyond permutation branching programs: Proofs, attacks, and candidates. In *Annual International Cryptology Conference*, pages 577–607. Springer, 2018.

[44] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new clt multilinear map over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.

[45] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 3–12, 2015.

[46] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 2016.

[47] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[48] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Trans. Information Theory*, 46(3):893–910, 2000.

[49] Clifford Cocks. An identity based encryption scheme based on Quadratic Residues. In *Cryptography and Coding, IMA International Conference*, volume 2260 of LNCS, pages 360–363, 2001.

[50] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, 2016.

[51] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, 2015.

[52] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, 2016.

[53] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography,*

Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I, 2017.

[54] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[55] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014.

[56] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, 2015.

[57] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[58] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

[59] Yevgeniy Dodis and Yu Yu. Overcoming weak expectations. In *Theory of Cryptography*, pages 1–22. Springer, 2013.

[60] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Advances in Cryptology - CRYPTO 2017*

- *37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 537–569, 2017.

[61] Nico Döttling and Dominique Schröder. Efficient pseudorandom functions via on-the-fly adaptation. In *Annual Cryptology Conference*, pages 329–350. Springer, 2015.

[62] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 265–284, 2006.

[63] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: Efficient algorithms and hardness results. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 381–390, New York, NY, USA, 2009. ACM.

[64] Nico Dttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. TCC, 2017.

[65] Nelly Fazio, Antonio Nicolosi, and Duong Hieu Phan. Traitor tracing with optimal transmission rate. In *Information Security, 10th International Conference, ISC 2007, Valparaíso, Chile, October 9-12, 2007, Proceedings*, pages 71–88, 2007.

[66] Amos Fiat and Moni Naor. Broadcast encryption. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 480–491, 1994.

[67] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 44–61, Berlin, Heidelberg, 2010. Springer-Verlag.

[68] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[69] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indstinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[70] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 121–130, New York, NY, USA, 2010. ACM.

[71] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, 2015.

[72] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[73] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[74] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. In *STOC*, 2013.

[75] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

[76] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[77] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Annual Cryptology Conference*, 2015.

[78] Rishab Goyal, Sam Kim, Nathan Manohar, Brent Waters, and David J. Wu. Watermarking public-key cryptographic primitives. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 367–398, 2019.

[79] Rishab Goyal, Venkata Koppula, Andrew Russell, and Brent Waters. Risky traitor tracing and new differential privacy negative results. In *Annual International Cryptology Conference*, pages 467–497. Springer, 2018.

[80] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[81] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In *EUROCRYPT*, 2017.

[82] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 660–670. ACM, 2018.

[83] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. *SIAM Journal on Computing*, (0):STOC18–94, 2019.

[84] Rishab Goyal, Venkata Koppula, and Brent Waters. New approaches to traitor tracing with embedded identities. In *TCC (to appear)*, 2019. https://eprint.iacr.org/2019/980.

[85] Rishab Goyal, Willy Quach, Brent Waters, and Daniel Wichs. Broadcast and trace with n^ϵ ciphertext size from standard assumptions. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 826–855, 2019.

[86] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, 2006.

[87] Shai Halevi. Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866, 2015.

[88] Nicholas Hopper, David Molnar, and David A. Wagner. From weak to strong watermarking. In *TCC*, 2007.

[89] Yupu Hu and Huiwen Jia. Cryptanalysis of ggh map. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.

[90] Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 450–465, 2002.

[91] Sam Kim and David J. Wu. Collusion resistant trace-and-revoke for arbitrary identities from standard assumptions. Cryptology ePrint Archive, Report 2019/984, 2019. https://eprint.iacr.org/2019/984.

[92] Lucas Kowalczyk, Tal Malkin, Jonathan Ullman, and Daniel Wichs. Hardness of non-interactive differential privacy from one-way functions, 2018.

[93] Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 145–157, 1998.

[94] Kaoru Kurosawa and Takuya Yoshida. Linear code implies public-key traitor tracing. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, pages 172–187, 2002.

[95] San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-lwe and applications in traitor tracing. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 315–334, 2014.

[96] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[97] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, April 2007.

[98] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13. In *Annual Cryptology Conference*, 2016.

[99] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *PKC*, 1999.

[100] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology - CRYPTO 2001*, 2001.

[101] Moni Naor and Benny Pinkas. Threshold traitor tracing. In *Advances in CryptologyCRYPTO'98*, pages 502–517. Springer, 1998.

[102] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In *Financial Cryptography, 4th International Conference, FC 2000*, 2000.

[103] Ryo Nishimaki. How to watermark cryptographic functions. In *EUROCRYPT*, 2013.

[104] Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 388–419, 2016.

[105] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342, 2009.

[106] Duong Hieu Phan, Reihaneh Safavi-Naini, and Dongvu Tonien. Generic construction of hybrid public key traitor tracing with full-public-traceability. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 264–275, 2006.

[107] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.

[108] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[109] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[110] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 463–472, New York, NY, USA, 2010. ACM.

[111] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[112] Amit Sahai and Brent Waters. Slides on functional encryption. PowerPoint presentation, 2008. http://www.cs.utexas.edu/~bwaters/presentations/files/functional.ppt.

[113] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014.

[114] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[115] Jessica Staddon, Douglas R. Stinson, and Ruizhong Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Information Theory*, 47(3):1042–1049, 2001.

[116] Douglas R. Stinson and Ruizhong Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discrete Math.*, 11(1):41–53, 1998.

[117] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 600–611, 2017.

[118] Andrew Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.

[119] Maki Yoshida and Toru Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1), 2011.